NASA Contractor Report 178174

ICASE REPORT NO. 86-58

ICASE

DYNAMIC REMAPPING DECISIONS IN MULTI-PHASE PARALLEL COMPUTATIONS

(NASA-CR-178174) DYNAMIC REMAPPING DECISIONS IN MULTI-PHASE PARALLEL COMPUTATIONS Final Report (NASA) 48 p
CSCL 09B G3/61

N87-10719
Unclas
44220

David M. Nicol

Paul F. Reynolds, Jr.

Contract Nos. NAS1-17070, NAS1-18107

September 1986

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

Dynamic Remapping Decisions in Multi-Phase Parallel Computations

David M. Nicol
Institute for Computer Applications in Science and Engineering

Paul F. Reynolds, Jr.
University of Virginia

Abstract The effectiveness of any given mapping of workload to processors in a parallel system is dependent on the stochastic behavior of the workload. Program behavior is often characterized by a sequence of phases, with phase changes occurring unpredictably. During a phase, the behavior is fairly stable, but may become quite different during the next phase. Thus a workload assignment generated for one phase may hinder performance during the next phase. We consider the problem of deciding whether to remap a parallel computation in the face of uncertainity in remapping's utility. Fundamentally, it is necessary to balance the expected remapping performance gain against the delay cost of remapping. This paper treats this problem formally by constructing a probabilistic model of a computation with at most two phases. We use stochastic dynamic programming to show that the remapping decision policy which minimizes the expected running time of the computation has an extremely simple structure: the optimal decision at any step is followed by comparing the probability of remapping gain against a threshold. This theoretical result stresses the importance of detecting a phase change, and assessing the possibility of gain from remapping. We also empirically study the sensitivity of optimal performance to imprecise decision thresholds. Under a wide range of model parameter values, we find nearly optimal performance if remapping is chosen simply when the gain probability is high. These results strongly suggest that except in extreme cases, the remapping decision problem is essentially that of dynamically determining whether gain can be achieved by remapping after a phase change; precise quantification of the decision model parameters is not necessary.

This research was supported in part by the National Aeronautics and Space Administration under NASA Contracts NAS1-17070 and NAS1-18107 while the first author was in residence at ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665. It was also supported in part by the Virginia Center for Innovative Technology, while both authors were in residence at the University of Virginia, Department of Computer Science, Thornton Hall, Charlottesville, VA 22903.

1. Introduction

An important issue in parallel processing is the assignment of workload to processors. A common model of this problem is to assume that a program is composed of a number of communicating modules, and that each module is to be assigned to a processor in the target parallel system. The assignment algorithm takes a global view of the system, and must consider processors' capacity, any special affinity a module has for a processor (e.g. a module may require assignment to a processor with a floating point accelerator), module execution requirements, inter-module communication, and any access to files and data structures that a module may require. The assignment algorithm may simultaneously assign files to different storage devices, so we will speak of the *mapping* of the computation, rather than just the module assignment. Any reasonable mapping algorithm must take into account the expected behavior of the mapped computation, because the efficiency of a parallel computation depends heavily on how well its mapping both exploits available parallelism, and minimizes the communication and synchronization overhead. Both of these factors are determined by the underlying stochastic behavior of the computation. If during run-time the anticipated behavior changes and causes a mismatch between mapping and behavior, performance will deteriorate. In this case, it can be desirable to dynamically remap the computation. Because of the complicated considerations often involved in task and file assignment, it may not be feasible to allow processors to move modules, files, and data structures around in a dynamic decentralized fashion. A global mapping (or remapping) algorithm is better able to consider all aspects of the assignment problem, especially if the parallel system is tightly coupled.

Another type of workload assignment is sometimes employed for parallel scientific computations. These computations are often composed of numerical calculations at each point in a discretized spatial (or transformed) domain. Workload assignment in this context involves partitioning the discretized domain points into *regions* which are then mapped to processors [2]; usually the number of regions

equals the number of processors. A processor's workload is then a function of the domain points in the region it receives. While the region partitioning is usually concerned with balancing the execution workload in each region, the assignment of regions to processors is mostly concerned with the communication costs incurred by the assignment. To dynamically remap the computation, we repartition the problem domain and assign the resulting regions to processors. Remapping might be desired if the number of domain points in a processor's region changes, or if the calculations required at the domain points change. A good example of this phenomenon is the behavior of shock-capturing techniques in computational gas dynamics [4]; the behavior change occurs when a shock develops and the numerical technique attempts to resolve the shock features with additional grid points. Since the domain was originally partitioned by physical region, the additional grid points can create a workload imbalance which leads to a performance decline. Many other numerical techniques adaptively change the grid in response to solution behavior and may thus exhibit phase-like behavior, for examples see [1].

In both of the fore-mentioned types of computation, we can expect run-time behavior to change unexpectedly, and to the detriment of run-time performance. Dynamic remapping might improve performance, but remapping raises a number of issues including (1) whether to use global remapping or decentralized and localized remapping, (2) determining that a phase change leading to potential performance gain has occurred, (3) determining a new mapping and its implementation, (4) determining the gains and costs of remapping, (5) determining the performance loss of not remapping after a phase change, and (6) optimally chosing when to remap. This paper treats only the latter issue, which must essentially balance all costs, gains, and uncertainities involved in the decision to remap. All of the other issues are likely to be problem and system dependent, and are interesting research issues in their own right. We focus on the remapping decision problem because it brings whatever solutions are found for all of the other issues (aside from (1)) together in a cohesive bundle. We are furthermore motivated to treat the decision problem mathematically, because a mathematical model can attempt to abstract the salient features of many diverse remapping situations. Furthermore, our model allows us

identify an optimal remapping decision policy for a two-phase model. Optimal model performance gives us a baseline we use to study the sensitivity of model performance to the non-estimation of remapping's costs and gains. The somewhat surprising conclusion of our sensitivity study is that for the purposes of deciding when to remap, the issues of estimating remapping's gains and losses are often not issues at all. Careful use of a formal decision model in real situations would require careful estimation of quantities which are difficult to predict a priori, e.g. post phase change performance degradation, post remapping performance gain, and gain detection accuracy. Our empirical study shows that our model achieves nearly optimal performance by simply remapping when the likelihood of achieving better performance after remapping is high. This implies that careful estimation of major decision model parameters is not necessary. We also show that this decision heuristic is effective when the number of phase changes is not limited to two. Of course, the applicability of this result to real remapping situations depends on the degree to which the situation matches the model; nevertheless, the result suggests that the dynamic remapping decision problem is manageable.

We have argued that some computations can benefit from dynamic global remapping. Most of the related literature does not address this particular problem. The work reported in [6], [7], [9], and [25] essentially presumes that jobs arrive at a central dispatcher which assigns jobs to processors. Our problem eschews the job arrival model, and does not allow a dynamic routing mechanism. A more recent body of work including [8], [13], [22], [23], [26] allows decentralized assignment decisions to be made dynamically. Again, our problem presumes that incremental dynamic reassignment is not feasible. Static and dynamic task assignment algorithms are presented in [2], [3], [5], [10], [11], [12], [17], [24]. The dynamic assignment algorithms consider restricted classes of computations; the static assignment algorithms might be used in conjunction with a remapping decision policy if the statically assigned computations abruptly change behavior. In [16] we treat dynamic remapping of parallel computations whose behaviors change constantly, and gradually; we focus here on behavior which changes radically, and abruptly. This paper is an extension of earlier treatments of this problem in [14] and

[15]. Our approach is a variation on aspects of the broad treatment of change detection under uncertainty given in [18]. Our model modifies this analysis by using a different decision cost structure, and by assuming a random computation duration. Our two main contributions are (1) to apply Markov decision theory to the performance issue of *when* to reconfigure a workload distribution in a parallel processing environment; (2) to show empirically that nearly optimal performance can be achieved without quantifying important decision model parameters.

Section 2 describes the decision model we study, and identifies important functions related to the remapping problem. Section 3 discusses the optimal decision policy, showing that it is a threshold policy. The calculation and behavior of the optimal thresholds is also discussed. Section 4 reports the results of an empirical study which examined the relative performance of fixed-threshold decision heuristics. Section 5 presents our conclusions, and the Appendix treats analytic issues in detail.

2. Problem Model

Our application of decision process theory requires that we identify a sequence of *decision points* in time where the decision is made whether to remap. Accordingly, we assume that the computation of interest gives rise to a natural sequence of decision points. For example, the end of an iteration in an iterative numerical program is a natural decision point. Likewise, natural decision points could be found in an embedded real-time system which periodically calls monitoring tasks. We define a *cycle* to be the amount of computation performed between two decision points. The time required by the system to execute a cycle, a *cycle time*, is assumed to be random. While we could allow the mean cycle time to depend on the decision points defining the cycle, for simplicity we will assume that mean cycle times during a phase are identical. We do **not** assume that cycle time distributions are independent, nor identically distributed. Also for simplicity's sake we assume that at most one phase change will occur during the course of the computation. We let e_F be the mean cycle time achieved during the first phase. e_B is the mean cycle time achieved after the phase change, but while the original mapping is in

place. Performance declines after the phase change if $e_B > e_F$. e_R is the mean cycle time during the second phase, after a remapping. We implicitly assume that $e_R < e_B$, but make no assumptions regarding the relationship between e_R and e_F.

A natural way of measuring the amount of work performed by the computation is to enumerate the number of cycles executed. We let N denote this quantity, and allow N to be random. We also assume that N is bounded from above by a constant M, and that N has a decreasing failure rate function. The usefulness of this latter assumption is buried deeply in the proof of Lemma 1. Here we will simply state that a decreasing failure rate means that the quotient

$$\frac{Prob\{N = n\}}{Prob\{N \geq n\}}$$

is an increasing function of n. This intuitively means that the longer the computation continues, the more likely it is that termination occurs at the present cycle.

Our decision model describes two types of uncertainity in the remapping problem. We cannot be sure when (or if) we can gain better performance by remapping. This uncertainity is modeled by presuming that the occurrence of a phase change leading to potential remapping gain is random; the probability distribution of the cycle during which this occurs is assumed to have a constant failure rate ϕ. The second source of uncertainity is related, but is perhaps more subtle. At a decision point we will employ some (problem dependent) mechanism to test for remapping gain. This mechanism might look for a decline in processor utilizations, or it might be able to examine what code has recently been executed. Based on such examinations, the mechanism can give us some indication of whether a new mapping is called for, but we cannot be certain that the mechanism is absolutely reliable. It might prematurely report the possibility of gain, or it might fail to report an existing possibility of gain. This type of uncertainty is modeled by assuming that every invocation of the mechanism has a probability α of prematurely reporting possible gain, and a probability β of failing to existing possible gain.

At every decision point in the computation, a decision policy decides whether to remap. The decision algorithm will consult the gain testing mechanism described above, but is distinct from that mechanism. The decision to remap is based on a *gain probability* which is calculated as a function of the response received from the gain testing mechanism. The probability is initially zero; if the first gain test detects no possibility for gain, we have *evidence* that there is no immediate gain from remapping. But the probability of remapping gain can no longer be zero since it is possible for a phase change to have occurred during the first cycle, and it is possible for remapping gain to be achieved, and it is possible that the test mechanism failed to detect the potential gain. The true value of the gain probability in this case depends on the values of ϕ and β. Similar observations hold if potential gain is reported. Bayes' Theorem [21] gives us a mechanism for calculating this probability. In general, let p_n be the probability of remapping gain calculated at the *nth* decision point. Initially, $p_1 = 0$. Supposing that $p_{n-1} = p$, p_n is found by first calculating

$$\begin{aligned} p^*(p) &= p + \phi(1 - p) \\ &= (1 - \phi)p + \phi. \end{aligned} \tag{1}$$

$p^*(p)$ is interpreted as the probability that gain will be possible by step n, given that $p_{n-1} = p$. This probability is calculated at step n–1, and so cannot consider the gain detection mechanism's report at step n. The value of p_n depends on this report, and is calculated using Bayes' Theorem as follows. If potential gain is reported, p_n is given by $p^c(p)$:

$$p_n = p^c(p) = \frac{p^*(p)\cdot(1 - \beta)}{p^*(p)\cdot(1 - \beta) + (1 - p^*(p))\cdot\alpha}. \tag{2}$$

Given a negative indication of potential gain, p_n is defined by $p^{\bar{c}}(p)$:

$$p_n = p^{\bar{c}}(p) = \frac{p^*(p)\cdot\beta}{p^*(p)\cdot\beta + (1 - p^*(p))\cdot(1 - \alpha)}. \tag{3}$$

We will require one other related probability. Let $q^c(p)$ be the probability that the gain detection mechanism will report potential gain at step n, given that $p_{n-1} = p$. By conditioning on whether gain is

actually possible by step n, it is not difficult to see that

$$q^c(p) = p^*(p)(1-\beta) + (1-p^*(p))\alpha. \quad (4)$$

The probability of the mechanism reporting no gain at step n given $p_{n-1} = p$ is simply $q^{\bar{c}}(p) = 1 - q^c(p)$.

A decision to remap incurs some explicit costs. First, there is a delay cost of calculating the remapping. We will furthermore suppose that after calculating the new mapping it is possible to compare its performance with that of the old mapping. If gain is possible from remapping, it will be found to be superior, it otherwise will not. This test can help us avoid an unnecessary and potentially costly implementation of the new mapping. We therefore let D_T be the delay cost of calculating and testing a new mapping, and let D_I be the delay cost of actually implementing that mapping. Table I summarizes our decision model definitions.

Notation	Definition
n	Decision Step Number
N	Random number of decision steps
M	Upper Bound on N
N_n	N given $N \geq n$
$\hat{N}_n$	$E[N_n]$
e_F	Decision Interval Pre-Gain Execution Time, Original Mapping
e_B	Decision Interval Post-Gain Execution Time, Original Mapping
e_R	Decision Interval Post-Gain Execution Time, New Mapping
D_T	Delay to Calculate and Test New Mapping
D_I	Delay to Implement New Mapping
α	Gain Test False Alarm Error
β	Gain Test Missed Gain Error
ϕ	Time of Gain Failure Rate Probability
$p^*(p)$	Pre-Observation Probability of Gain At Next Decision Step
$p^c(p)$	Posterior Probability of Gain After Positive Gain Observation
$p^{\bar{c}}(p)$	Posterior Probability of Gain After Negative Gain Observation
$q^c(p)$	Probability of Observing Gain Next Observation
$q^{\bar{c}}(p)$	Probability of Not Observing Gain Next Observation

Table I

Every decision made by a remapping decision process incurs an explicit cost which reflects the expected cycle time of the cycle following the decision, and any remapping overhead costs. For example, if a phase change has occurred which allows remapping gain, but the old mapping is retained, then e_B is the expected cycle time of the next cycle. If remapping is chosen, but no gain is yet possible, an overhead cost of D_T is suffered, the absence of gain is discovered, and the old mapping is retained. If remapping is chosen when gain is achievable, then an overhead cost of $D_T + D_I$ is suffered, but then every remaining cycle has a mean execution time of e_R. The total computation execution time is the sum of all cycle times plus remapping overhead; an optimal decision policy minimizes the expectation of this sum. We see that the optimal decision policy should depend somehow on the various costs and gains involved in the remapping process, the remaining length of the computation, and the degree of our certainity that gain is possible. One way to express the inter-relationships between all of these concerns is as a stochastic dynamic programming problem. Given gain probability p at step n, let $V(<p,n>)$ denote the expected remaining execution time of the computation if we use the optimal decision policy. In the parlance of Markov decision processes [19], we are defining $<p,n>$ to be the state of the process at step n, and $V(<p,n>)$ to be the optimal (stationary) cost function. If we choose to retain the old mapping at step n, the next cycle's expected execution time is

$$p^*(p)e_B + (1 - p^*(p))e_F.$$

Note that this expression anticipates the possibility that the next cycle will be the first during which gain is possible; in this case the next cycle's mean execution time is assumed to be e_B. Let $E_v(<p,n>)$ be the expected remaining execution time after step n+1, using the optimal decision policy, given a gain probability p at step n and retention of the mapping at step n. Then the expected execution time remaining after step n, achieved by keeping the old mapping now and thereafter using the optimal decision policy is

$$C_t(<p,n>) = p^*(p)e_B + (1 - p^*(p))e_F + E_v(<p,n>). \tag{5}$$

We call C_t the *retain cost function.* We similarly define $C_m(<p,n>)$, the *remap cost function.*

$C_m(<p,n>)$ is the expected remaining execution time achieved by choosing to remap now, and thereafter using the optimal decision policy. By choosing to remap, we immediately incur an overhead cost D_T. If after testing the new mapping we find that gain is possible, then all remaining cycles have mean execution time e_R. There are an expected number $\hat{N}_n - n + 1$ of these cycles, where $\hat{N}_n$ denotes the expected value of N given that $N > n$. Furthermore, we consider the decision process to have stopped at this point, because the single possible change has occurred. If the new mapping is found to be no better than the old, then the old mapping is retained, the next cycle has mean execution time $p^*(0)e_B + (1 - p^*(0))e_F$, the probability of gain is set to zero, and the decision process continues. Thus we see that

$$C_m(<p,n>) = D_T + p\left[D_I + (\hat{N}_n - n + 1)e_R\right] + (1 - p)\left[p^*(0)e_B + (1 - p^*(0))e_F + E_v(<0,n>)\right]. \quad (6)$$

The function $E_v(<\cdot,n>)$ appears in the definition of both $C_t(<\cdot,n>)$ and $C_m(<\cdot,n>)$. Since it describes optimal decision policy costs after step n+1, it is stated in terms of $V(<\cdot,n+1>)$. $E_v(<\cdot,n>)$ is a function expressing expected values, taken with respect to the probability of reporting potential gain at step n+1. Given $p_n = p$, p_{n+1} will be equal to $p^c(p)$ if the mechanism at step n+1 reports potential gain, this will occur with probability $q^c(p)$. Similar observations apply in the event that no change is observed. Thus we see that

$$E_v(<p,n>) = q^c(p)V(<p^c(p),n+1>) + q^{\bar{c}}(p)V(<p^{\bar{c}}(p),n+1>).$$

Now the principle of optimality states that

$$V(<p,n>) = \min\begin{cases} C_m(<p,n>) \\ C_t(<p,n>) \end{cases},$$

and that the optimal decision at step n given $p_n = p$ is the decision whose cost function minimizes the right hand side of the equation above. $C_m(<\cdot,n>)$ and $C_t(<\cdot,n>)$ are both functions of $V(<\cdot,n+1>)$; to determine the optimal decision policy we need to solve equations recursing on n. Since the number of decision steps is bounded above by M, we can start the solution procedure by defining

$V(<p,M+1>) = 0$ for all $p \in [0,1]$, and then determine $V(<p,M>)$ for all $p \in [0, 1]$. An algorithm for computing $V(<\cdot,n>)$ in terms of $V(<\cdot,n+1>)$ is given in the Appendix. The following section shows that $V(<\cdot,n>)$ has a useful structure, and that the optimal decision from state $<p,n>$ is nicely characterized.

3. Optimal Decision Policy Thresholds

We have noted that the value of the gain probability is a key factor in our decision process. In this section we show that the optimal decision policy is a threshold policy: for every decision step n there is a threshold $\pi_n \in [0,1]$ such that the optimal decision in state $<p,n>$ is to remap if $p > \pi_n$, and retain if $p \leq \pi_n$. We then show why exact calculation of the thresholds $\{\pi_n\}$ is computationally intractable, report on the computational complexity of an approximation technique having bounded error, and graphically illustrate the behavior of the $\{\pi_n\}$ as a function of n.

The following lemma provides the fundamental reasons for the optimal policy structure. Its proof is somewhat lengthy, and is given in the Appendix.

LEMMA 1 :

- For all n, $C_m(<p,n>)$ is a linear function of p;
- For all n, $C_t(<p,n>)$ is a piecewise linear concave function of p;
- There exists $n_0 \in [0,\infty]$ such that for all $n \geq n_0$, $C_t(<p,n>) \leq C_m(<p,n>)$ for all $p \in [0,1]$;
- If $n < n_0$, $C_m(<1,n>) \leq C_t(<1,n>)$.

□

Consider the implications of Lemma 1. For any step $n \geq n_0$, the retain decision cost function is always less than the remap decision cost function, implying that we should retain regardless of the value of the gain probability. In this case, the optimal decision threshold is degenerate, $p_n = 1$. For $n < n_0$ we know that the linear remapping function is less than the concave retain function at $p = 1$. It is therefore geometrically impossible for $C_t(<\cdot,n>)$'s functional curve to intersect $C_m(<\cdot,n>)$'s functional curve more than once, as illustrated by figure 1. If $C_t(<\cdot,n>)$ and $C_m(<\cdot,n>)$ intersect at $\pi_n < 1$, then

$C_t(<p,n>)$ is less than $C_m(<p,n>)$ for $p \in [0,\pi_n]$, and $C_m(<p,n>)$ is less than $C_t(<p,n>)$ for $p \in [\pi_n,1]$. It follows that the optimal decision from state $<p,n>$ is to retain if $p \leq \pi_n$, and to remap if $p > \pi_n$. We summarize this result:

THEOREM 1 : For every step n, there exists a π_n such that the optimal decision from $<p,n>$ is to remap if and only if $p > \pi_n$.

□

If all the decision model parameters are known, then in theory we can solve the equations describing $V(<p,n>)$ and determine each optimal threshold. In practice there are significant obstacles to this procedure. We may not be able to quantify the model parameters; we defer this problem to section 4. A computational problem arises from the fact that the optimal cost functions $V(<\cdot,n>)$ are all piece-wise linear. If a piece-wise linear function changes its linear description at domain point d, we will call d a *transition* point. For any piece-wise linear function g on [0, 1], let $D(g)$ be the set of g's transition points. In [14] we show that

$$D(C_t(<\cdot,n-1>)) = \{q \geq 0 \mid p^c(q) = d \text{ or } p^{\bar{c}}(q) = d \text{ for some } d \in D(V(<\cdot,n>)\}.$$

This means that every transition point for $V(<\cdot,n>)$ can give rise to two distinct transition points for $C_t(<\cdot,n-1>)$. Any of these transition points greater than π_{n-1} will not appear in $D(V(<\cdot,n-1>))$; nevertheless, we see that the number of line segments defining $V(<\cdot,n>)$ essentially doubles at every step of the recursive solution. Then in general, an exact solution is not computationally feasible. However, in the Appendix we describe an approximation procedure which estimates $V(<\cdot,n>)$ to any desired degree of accuracy. Furthermore, at every step this procedure is linear in the number of transition points, and over a broad class of approximations, it minimizes the number of transition points required to achieve the desired accuracy. Our computational experience with this approximation shows that it is quite robust. With the scale of parameter values we used, generally fewer than 200 transition points in the approximation bounded the error in approximating $V(<\cdot,n>)$ for each n by 10^{-5}.

Figure 2 illustrates the behavior of the optimal thresholds for four different sets of parameter values. For computational simplicity, the number of cycles in the computation was assumed to be constant rather than random. In our experience, we have always seen the tendency for the optimal thresholds to remain relatively constant, except for n nearest N. Note also that as the per cycle gain $G = e_B - e_R$ increases (for fixed costs D), the converged value of the optimal threshold decreases. Intuitively this is true because the smaller the gain from remapping is, the more certain we should be that gain is possible before choosing remapping and suffering the attendant overhead. Another tendency is that the region where $\pi_n = 1$ increases as the remapping overhead costs $D = D_T + D_I$ increase. This too makes sense, because the expected overall remapping gain depends on the expected remaining length of the computation. If this is small, then a large remapping overhead may not be amortized, and it is better to simply suffer the "bad" cycle times e_B until termination.

4. Model Sensitivity

Calculation of the the optimal decision policy requires quantification of the model parameters. However, some of these parameters may be difficult to estimate at run-time. For example, it is unreasonable to assume that we can accurately predict the post-gain cycle execution time means e_R and e_B. Because of this problem, we examined the deviation of run-time performance from optimal performance if remapping is chosen whenever $p_n > \rho$, for some fixed ρ. This class of heuristics is suggested by the behavior of the optimal thresholds, as illustrated by figure 2: for most n, π_n is relatively constant for most of the computation's steps. Our experiments varied ρ between 0.05 and 0.95. This section reports the results of that study. We find that over a wide range of model parameter values, nearly optimal performance is achieved if ρ is chosen to be high, but not excessively so (e.g. $\rho \in [0.7, 0.8]$). From this data we conclude that the estimation of remapping costs and gains is not as critical a problem as we might otherwise suppose. We then examine the sensitivity of performance when we incorrectly calculate the gain probability. We still find that for limited degrees of inaccuracy in our

estimates of α, β, and ϕ, nearly optimal performance can be achieved. From this data we conclude that the critical issue in the remapping decision problem is to be able to determine when some kind of gain might be possible by remapping. Finally, we note that the decision heuristic extends in a natural way to computations with potentially more than two phases, and see again that the heuristic is effective.

Our empirical study used a simulation of the analytic model we have already discussed. For every set of model parameters, the optimal decision thresholds were approximated with high accuracy. Then for each $\rho = m \cdot 0.05$, $m = 1,2,...,19$, 1000 simulation runs of the modeled system were performed, tabulating the run-time achieved under the optimal policy, and the policy which remaps whenever the gain probability exceeds ρ. The run-time achieved by never remapping was also tabulated. For each parameter set and for both non-optimal policies, the relative difference between the non-optimal policy performance and the optimal policy performance was calculated. We graphically mapped this relative difference as a function of ρ.

In doing a sensitivity study we are confronted with the problem of finding an appropriate collection of parameter sets. It is intuitively clear that remapping can be useful only if its gains are not dominated by its costs; we first constrain the space of parameter values by estimating an envelope of parameter values where gains exceed costs. The envelope is calculated as a function of N, ϕ and $D = D_T + D_I$ as follows. First, we calculate the expected possible gain. The per cycle gain is $G = e_B - e_R$; for constant N, the expected possible gain is simply

$$\begin{aligned} E_G &= \sum_{i=1}^{N} G \cdot (N - i) \cdot Prob\{\text{gain first achieved at } i\} \\ &= G \sum_{i=1}^{N} (N - i) \cdot \phi (1 - \phi)^{i-1} \\ &= G \left[N - \frac{1 - (1 - \phi)^N}{\phi} \right]. \end{aligned}$$

The last step follows from uninteresting algebra which uses the fact that the time of gain distribution is

nearly a geometric distribution. Assuming that the remapping overhead consists only of one successful remapping test and implementation, the envelope of admissible gain and cost parameters is found by equating E_G to $D_T + D_I$. That is, for any value of G, any remapping cost up to E_G is less than the expected possible gain. Figure 3 shows the envelope under the assumptions that $N = 100$, $\phi = 3/100$, and $e_B = 200$ (which bounds G from above by 200). We will use these parameter values throughout our study; in addition, we assume that $D_T = D_I$, and denote $D = D_T + D_I$. The dashed lines in figure 3 identify the subsets of the envelope space where we will choose G and $D_T + D_I$ for our empirical study. These lines fairly well span the space of admissible cost and gain values, and should give us a reasonable picture of performance sensitivity throughout the envelope. Note that the dashed lines delineate the marginal functions of G and D from the three (G, D) coordinates (50,200), (100,3000), and (150,9000).

N was set to 100 in all of the simulations we report here. This seems reasonable since dynamic remapping is a consideration only for relatively long-lived computations, and simulations showed that except for small N ($N \leq 10$), N doesn't affect the performance measures much. Our studies looked at marginal performance with respect to G, to $D = D_T + D_I$, and to gain detection accuracy. Essentially, we varied these parameters as much as possible through the three interior points identified in figure 3. The studies which focused on costs and gains assumed that $\alpha = \beta = 0.25$.

Each graph maps the performance of heuristic policies in relation to the optimal policy. A data point is generated by 1000 runs of a simulator which calculates the execution time under (1) the optimal policy (2) a fixed ρ-threshold policy, and (3) never remapping, or the NR policy. If the sum of execution times for policy h is *hcost* and the sum of execution times for the optimal policy is *ocost*, then the data point plotted for policy h is $(hcost - ocost)/ocost$. The independent variable in each of these graphs is ρ. The piece-wise linear curves map the relative difference between the ρ-threshold policy and optimal policy as a function of ρ. Each strictly horizontal curve plots the relative difference

between the NR and optimal policies.

Figures 4,5, and 6 show the sensitivity to changing G through the envelope coordinates (50,200), (100,3000), and (150,9000) respectively. Every ρ-threshold policy and NR policy curve is marked with the G value defining that curve. On each of the graphs we notice that the relative performance of the NR policy deviates from the optimal sharply as we increase G. This comes as no surprise, since optimal performance is achieved when there is no possible gain by simply never remapping. For every fixed value of G, it is also interesting to compare the relative performance of the NR and ρ-threshold policies, by looking for the point of intersection between their respective curves. For example, we find no such intersections in figure 4, implying that within the indicated parameter range, any ρ-threshold policy will yield better performance than the NR policy. This is not the case in figures 5 and 6 where remapping costs are much higher. There we have marked the point of intersection for each pair of curves associated with a fixed value of G. Every pair of curves shown in figure 5 have a point of intersection, the largest is approximately $\rho = 0.5$ and is associated with $G = 50$. For the range of parameter values given by this graph, we see that choosing $\rho > 0.5$ leads to performance better than the NR policy. The actual performance gain over the NR policy depends strongly on the value of G. Similar conclusions can be drawn from Figure 6. The curves for $G = 150$ intersect at approximately $\rho = 0.65$; for any $G \geq 150$ and any $\rho > 0.65$, we can outperform the NR policy. On the other hand, the curves for $G = 100$ do not intersect, and the NR policy outperforms every ρ-threshold policy. Note too that the NR policy in this case is nearly optimal, and that the point (100,9000) lies outside of the envelope. Overall, the deviation from optimality of the ρ-threshold policy for high ρ is generally less than 10%, and can be less than 2-3%.

Figures 7,8 and 9 illustrate the sensitivity to changing D. We can make observations similar to those on changing G. As the cost increases, the difference between the optimal policy and the NR policy decreases. The points of intersection between policy curves are again marked, and again we see

that choosing $\rho > 0.65$ leads to performance gains over the NR policy. Once again we see that the performance of the ρ-threshold policy for high ρ deviates only slightly from the performance of the optimal policy. Any non-monotonicity in the ρ-threshold curves are likely due to statistical fluctuation.

Figures 10, 11, and 12 show the sensitivity to changing gain test accuracy. For simplicity we assumed that $\alpha = \beta$, and varied these parameters from 0.05 to 0.5. No larger values of these parameters need be considered, since the information content of a test with error probability p is equal to that of a test with error probability $1 - p$. Collectively, these figures show that optimal performance is relatively less sensitive to gain test accuracy than it is to remapping cost or gain. Figure 10 shows the sensitivity at (50,200), where we see that if $\alpha \leq 0.4$, then every ρ-threshold policy achieves better performance than the NR policy. The case where $\alpha = 0.5$ provides an interesting contrast, where if ρ is too *high*, then the NR policy is slightly better. This curious effect is understood by realizing that a test with 50% failure gives no information at all. The only evidence the model receives for possible gain is from the time of gain distribution. A high probability of gain is achieved only at steps near the end of the computation; but then the costs of remapping threaten to dominate the gains. This same phenomenon is observed when $\alpha = 0.5$ in figures 11b and 12b. The sensitivity at (100,3000) and (150,9000) is more easily discerned by considering high accuracy and low accuracy cases separately. Figures 11a and 12a show the high accuracy cases. Once again we illustrate the points of policy curve intersection, and again we see that simply choosing a relatively high value of ρ leads to performance gains over the NR policy. This is also true for low accuracy values of α at (100,3000) (with the exception of $\alpha = 0.5$) shown in figure 11b, but is not the case at (150,9000). Figure 12b clearly shows that at (150,9000), low accuracy is bad news. Only in the case of $\alpha = 0.3$ can we outperform the NR policy, and then only marginally. At the envelope boundary we apparently need relatively high accuracy gain detection tests (figures 6 and 9 show that $\alpha = \beta = 0.25$ is adequate). Again, locally non-monotone behavior in the ρ-threshold curves is likely due to statistical fluctuation.

We have so far assumed that the ρ-threshold heuristic knows the precise values of α, β and φ. Since this is not likely to be true in practise, we examined the effect of using erroneous values for these parameters. Figure 13 shows the sensitivity of the ρ-threshold policy to faulty values of α and β. Once again, we calculated the relative difference from optimal performance (when α, β, and φ are precisely known). The measurements were taken when $G = 100$, $D = 3000$, and the true value of $\alpha = \beta$ is 0.25. A curve labeled with scale factor f denotes the performance achieved when the heuristic used $f \cdot 0.25$ as α (and β). As before, the horizontal line illustrates the relative performance of the NR policy. Figure 13 shows that even if we assume that the gain detection mechanism yields no information (scale factor of 2), that we can outperform the NR policy. It is also clear though that serious mis-estimation of α and β leads to significant performance decline.

Figure 14 looks at the sensitivity to faulty values of φ. The measurements were taken with the same parameter values as assumed in figure 13. Again, each curve is labeled with a factor used to scale φ up or down for the ρ-threshold heuristic. Here we see that the consequences of over-estimating φ can be quite serious; this arises because a high value of φ increases the gain probability, leading to an increase in the number of premature remapping decisions. At first glance, it seems counter-intuitive that strictly under-estimating φ should be better than using an exact value of φ. This phenomenon is understood by noting that the ρ-threshold curve with exact φ at envelope point (100,3000) is monotone decreasing (at least for $\rho \leq .95$). The net effect of underestimating φ is to hamper the increase in gain probability after positive gain test results, meaning that more post-gain cycles will occur before the gain probability is high enough to exceed the chosen threshold. For exact φ, this same effect is obtained by increasing the threshold value. Thus, we may think of the curves obtained by underestimating φ as left-moving translations of the exact φ curve. Since the exact φ curve is monotone decreasing, the ρ-threshold policy which underestimates φ will perform better than the ρ-threshold policy which uses an exact value of φ. If φ is unknown, it appears to be best to adopt a low value of φ, and let the statistical weight of successive positive gain tests drive the gain probability up.

As a final note, all of the studies we have reported are for the two phase model. It is likely that the optimal policy for a multiple (more than 2) phase model will have a threshold structure, but in view of the limited utility and added complexity of determining the optimal policy, we will not attempt to prove this. We did however compare a ρ-threshold heuristic with the NR policy on a multi-phase model. At every phase change, the model increased the non-remapped cycle execution time by 50 units, and increased the remapping gain by 25 units. We used $\rho = 0.75$, ϕ was set to 0.05, N was set to 250, and we assumed that $\alpha = \beta = 0.25$. The results were quite encouraging. Starting at the envelope point (150,9000) and $e_B = 200$, the 0.75-threshold policy outperformed the NR policy by a relative percentage of 66%. From (100,3000) the 0.75-threshold policy outperformed the NR policy by a relative percentage 202%. It is clear then that for multi-phase computations, a simple ρ-threshold policy can significantly improve performance.

We can draw several important conclusions from our sensitivity study. ρ-threshold policies work quite well when ρ is relatively high. In these cases, a ρ-threshold policy will almost always outperform the NR policy; furthermore, we can expect its performance to be within a few percentage points of the optimal policy. It is clear that we need to be careful about extremely high remapping costs, and inaccurate gain detection mechanisms. To protect ourselves from an inaccurate gain detection mechanism we can choose ρ so that it is not too high. Apparently a value of $\rho \in [0.7, 0.8]$ is high enough to protect against cost/gain imbalance, and low enough to protect against inaccurate gain detection (be warned, however, that this latter protection depends on ϕ). The most important implication of all this is that accurate estimation of per cycle remapping gain G is not necessary, nearly optimal performance can be achieved with a fixed threshold policy. We should try to determine whether the relationship between the per cycle gain G and remapping costs D allows remapping, but mapping costs can be surprising high and still allow remapping gain. For example, the ratio of remapping cost to per cycle gain with a gain of 150 and cost of 9000 is 60. In general, the largest permissible such ratio depends on N and ϕ; but with a relatively long computation, and probability of phase change exceeding 1/2, we

can expect a high ratio. We have also seen that a certain degree of inaccuracy in estimating α, β and ϕ can be tolerated. This is important, as these parameters too are not likely to be known exactly. Finally, we saw that it is reasonable to expect good performance from threshold policies used on computations with multiple phase transitions, provided that the phases are relatively long-lived.

5. Conclusions

An effective mapping of workload to processors in a parallel processing system must make certain assumptions about the computation's running behavior. The behavior of many computations are characterized as a sequence of phases, where behavior within a phase is fairly stable, but the behavior between two phases can be quite different. It is therefore possible for a mapping to become ineffective when a phase change occurs, so that dynamically *remapping* the computation may be required to maintain good performance. However the decision to remap must take into account the performance gains and costs involved, and must deal with uncertainity in whether remapping leads to gains. We have modeled this decision problem with a Markov decision process, and have determined the structure of the optimal decision policy. While this is an interesting theoretical result, it is not immediately practical. We therefore empirically studied the performance of a simple threshold heuristic which do not assume knowledge of remapping's costs and gains. We found that this heuristic works remarkablely well, implying that the remapping decision problem does not require precise estimates of these parameters. The key issue for the remapping decision problem is thus the relatively accurate assessment of when remapping leads to performance gains.

Appendix

In this appendix we prove Lemma 1 from section 3, and discuss an algorithm for approximating $V(<\cdot,n>)$. We first prove Lemma 1.

Proof of Lemma 1

Some of our analysis conditions on the value of N. We use the notation $g(<p,n>|N=m)$ to denote the value of function g at state $<p,n>$ given that $N = m$. We will also say that a function g is *plcc* if it is piece-wise linear, continuous, and concave.

Lemma 1's first claim is that for every n, $C_m(<p,n>)$ is a linear function of p. This is easily seen from its definition in equation (6); note that $E_v(<0,n>)$ does not depend on p. Lemma 1's second claim is that for every n, $C_m(<p,n>)$ is a *plcc* function of p. This result follows primarily from the following lemma reported in [18] and stated in terms of our notation:

LEMMA A-1 : Suppose that $N = m$. If $V(<p,n+1>|N=m)$ is a *plcc* function of p, then $E_v(<p,n>|N=m)$ is a *plcc* function of p.
□

We use this lemma to establish Lemma 1's second claim, by showing that for every fixed $n \geq 0$, $V(<p,n>)$ and $C_t(<p,n>)$ are *plcc* functions of p. First condition on $N = m$ for some m. We will inductively show that $V(<p,n>|N=m)$ and $C_t(<p,n>|N=m)$ are *plcc* functions of p. For the base case we consider $n = m$. For any $p \in [0,1]$,

$$C_t(<p,m>|N=m) = p^*(p)e_B + (1 - p^*(p))e_F + E_v(<p,m>|N=m)$$

$$= p^*(p)e_B + (1 - p^*(p))e_F$$

since $V(<p,m+1>|N=m) = 0$ for all p. Since $p^*(p)$ is linear in p, $C_t(<p,m>|N=m)$ is also linear in p. We also observe that $C_m(<p,m>|N=m)$ is *plcc* since it is linear. The class of *plcc* functions is closed under the point-wise minimum operation; $V(<p,m>|N=m)$ must also be *plcc*, establishing the induction base.

For the induction hypothesis we suppose that both $C_t(<p,n+1>|N=m)$ and $V(<p,n+1>|N=m)$ are *plcc* functions of p for some $n \leq m-1$. Lemma A-1, and the closure of *plcc* functions under addition and point-wise minimum again ensure that $C_t(<p,n>)$ and $V(<p,n>)$ are *plcc* functions of p, completing the induction.

To complete the proof, we note that the class of *plcc* functions is also closed under scalar multiplication, and observe that

$$V(<p,n>) = \sum_{m=0}^{M} Prob\{N = m\} \cdot V(<p,n>|N=m)$$

and

$$C_t(<p,n>) = \sum_{m=0}^{M} Prob\{N = m\} \cdot C_t(<p,n>|N=m)$$

To help establish Lemma 1's third and fourth claims, we analyze the values of $C_m(<p,n>)$ and $C_t(<p,n>)$ at $p = 1$. Key results are given by Lemma A-2.

LEMMA A-2 : Either

(i) $V(<1,n>) = C_m(<1,n>)$ for all n for which $Prob\{N = n\} \neq 0$; or

(ii) $V(<1,n>) = C_t(<1,n>)$ for all n for which $Prob\{N = n\} \neq 0$; or

(iii) There exists an n_0 (possibly ∞) such that for all $n < n_0$, $V(<1,n>) = C_m(<1,n>)$, and for all $n \geq n_0$ for which $Prob\{N = n\} \neq 0$, $V(<1,n>) = C_t(<1,n>)$.

PROOF: We condition on $N = m$, for any $0 \leq m \leq M$. Let K be the largest integer such that $(e_F - e_R) \cdot K \leq D_T + D_I$. Simple algebra (omitted here) establishes the inductive proof that for all n such that $m - K < n \leq m$,

$$V(<1,n>|N=m) = C_t(<1,n>|N=m) = (m - n + 1)e_B;$$

and that for $0 \leq n \leq m - K$,

$$V(<1,n>|N=m) = C_m(<1,n>|N=m) = D_T + D_I + (m - n + 1)e_R$$

and

$$C_t(<1,n>|N=m) = e_B + D_T + D_I + (m - n)e_R.$$

Define $d(n|N=m)$ to be the conditional difference $C_m(<1,n>|N=m) - C_t(<1,n>|N=m)$, and $d(n)$ to be the unconditional difference $C_m(<1,n>) - C_t(<1,n>)$. From the equations above, we see that as a function of m,

$$d(n|N=m) = \begin{cases} D_T + D_I - (e_B - e_R)\cdot(m - n + 1) & \text{for } m < n + K \\ (e_R - e_B) & \text{for } n + K \le m \end{cases}.$$

It follows from the definition of K that $d(n|N=m)$ is a decreasing function of m. The unconditional difference $d(n)$ is obtained by taking the expectation of $d(n|N=m)$ with respect to the residual distribution N_n of N given $N \ge n$. In [20] it is shown that if N has a decreasing failure rate function, then $E[g(N_n)] \le E[g(N_{n+1})]$ for all decreasing functions g. In particular, $d(n) \le d(n+1)$, showing that the difference $C_m(<1,n>) - C_t(<1,n>)$ is an *increasing* function of n. Then case (i) occurs if $d(n)$ is negative for all n, case (ii) occurs if $d(n)$ is positive for all n, and case (iii) occurs if $d(n)$ changes sign at $n = n_0$.

□

These results establish Lemma 1's fourth claim, that if $n < n_0$, then $C_t(<1,n>) \ge C_m(<1,n>)$. Finally, to establish its third claim, we will show that $C_t(<p,n>)$ is linear in p whenever $n \ge n_0$. Since $C_m(<\cdot,n>)$ is always linear, and $C_t(<0,n>) \le C_m(<0,n>)$ for all n, and $C_t(<1,n>) \le C_m(<1,n>)$ when $n \ge n_0$, it will follow directly that C_t and C_m cannot intersect, so that $C_t(<p,n>) \le C_m(<p,n>)$ for all $p \in [0, 1]$.

LEMMA A-3 : If $n \ge n_0$, then $C_t(<p,n>)$ is linear in p, and $V(<p,n>) = C_t(<p,n>)$ for all $p \in [0,1]$.

PROOF: We proceed by induction. M is the largest integer such that $Prob\{N = M\} \ne 0$, so that $V(<p,M+1>) = 0$ for all p and

$$V(<p,M>) = \min \begin{cases} D_l + p(e_R + D_r) + (1-p)(p^*(0)e_B + (1-p^*(0))e_F) \\ p^*(p)e_B + (1-p^*(p))e_F \end{cases}.$$

Presuming that $n_0 < M$, we have $C_t(<1,M>) \le C_m(<1,M>)$ and $C_t(<0,M>) \le C_m(<0,M>)$, so that $V(<p,M>) = C_t(<p,M>) = p^*(p)e_B + (1-p^*(p))e_F$, which is linear in p. The induction base is thus satisfied.

For the induction hypothesis, we suppose there is an $n > n_0$ such that $V(<p,n+1>) = C_t(<p,n+1>)$ for all $p \in [0,1]$, and that $C_t(<p,n+1>)$ is linear in p. Equation (5) implies that

$$C_t(<p,n>) = p^*(p)\cdot e_B + (1-p^*(p))e_F + q^c(p)\cdot V(<p^c(p),n+1>) + q^{\bar{c}}(p)\cdot V(<p^{\bar{c}}(p),n+1>),$$

and the induction hypothesis states that

$$V(<p,n+1>) = A\cdot p + B$$

for some A and B. Equations (2), (3), and (4) show that that $p^c(p) = \dfrac{p^*(p)(1-\beta)}{q^c(p)}$ and that $p^{\bar{c}}(p) = \dfrac{p^*(p)\beta}{q^{\bar{c}}(p)}$; it follows that

$$C_t(<p,n>) = p^*(p)e_B + (1-p^*(p))e_F + Ap^*(p) + B$$

which is linear in p since $p^*(p)$ is linear in p. Since $C_m(<p,n>)$ and $C_t(<p,n>)$ cannot intersect it follows directly that $C_m(<p,n>)$ exceeds $C_t(<p,n>)$ for all $p \in [0, 1]$. Thus $V(<p,n>) = C_t(<p,n>)$, completing the induction.

□

Calculating/Approximating $V(<\cdot,n>)$

We now discuss an algorithm for calculating or approximating the functions $V(<\cdot,n>)$. The proofs for various properties we claim for this algorithm are detailed in [14].

To deal with the fact that N can be random, we will condition on $N = m$ for some m. The procedure we describe is then repeated for all m such that $Prob\{N = m\} \neq 0$, and then the conditional functions are combined.

We know that $V(<\cdot,n>|N=m) = C_t(<\cdot,n>|N=m)$ is a linear function for $n \geq n_0$. It is a simple matter to define $V(<p,m+1>|N=m) = 0$ for all p, and then determine the slope and intercept of C_t and C_m using equations (5) and (6). Since $C_t(<0,n>|N=m)$ is less than $C_m(<0,n>|N=m)$ for all n, we test for an intersection of C_t and C_m by checking to see if $C_t(<1,n>|N=m) > C_m(<1,n>|N=m)$. If $C_t(<p,m>|N=m)$ and $C_m(<p,m>|N=m)$ do not intersect, we infer that $V(<p,m>|N=m) = C_t(<p,m>|N=m)$, and calculate the slope and intercepts for $C_t(<\cdot,m-1>|N=m)$ and $C_m(<\cdot,m-1>|N=m)$; again, examining the functional values at $p = 1$ will determine whether intersection has occurred. We repeat this process until the two cost curves intersect.

Now we assume that $V(<\cdot,n>|N=m)$ or some approximation to $V(<\cdot,n>|N=m)$ is known. A convenient representation for this function is as an array of records where each record holds a transition point, the value of V at that point, and the slope and intercept of the linear segment extending to the value of the function at the next greatest transition point. This array is sorted by the transition point values. The set of transition points for $C_t(<\cdot,n-1>|N=m)$ is found by defining the sets

$$S_1 = \{q \geq 0 \mid p^c(q) = d \text{ for some } d \in D(V(<\cdot,n>))\}$$

and

$$S_2 = \{q \geq 0 \mid p^{\bar{c}}(q) = d \text{ for some } d \in D(V(<\cdot,n>))\}$$

Since both $p^c(p)$ and $p^{\bar{c}}(p)$ are increasing functions of p (when $\alpha,\beta \leq 0.5$), sorted representations of S_1 and S_2 are easily obtained by simply choosing the d's for solution in $p^c(q) = d$ and $p^{\bar{c}}(p) = d$ in sorted order. The ordered lists for S_1 and S_2 are then merged (removing identical entries if necessary) into a sorted list for $D(C_t(<\cdot,n-1>|N=m))$. The function values for C_t at these points are then determined by equation (6). For any $e \in D(C_t(<\cdot,n-1>|N=m))$ the calculation of $C_t(<e,n-1>|N=m)$ requires

identification of $V(<p^c(e),n>|N=m)$ and $V(<p^{\bar{c}}(e),n>|N=m)$. This is efficiently done if we start the C_t value calculation procedure by first placing a "$p^c(\cdot)$–pointer" and a "$p^{\bar{c}}(\cdot)$–pointer" to the head of the $V(<\cdot,n>|N=m)$ array, and then process the points in $D(C_t(<\cdot,n-1>|N=m))$ in sorted order. To determine $V(<pc(e),n>|N=m)$ we simply advance the $p^c(\cdot)$ pointer until the appropriate linear segment of $V(<\cdot,n>|N=m)$ is encountered. Since $p^c(p)$ is increasing in p, the fact that the points in $D(C_t(<\cdot,n-1>|N=m))$ are processed in sorted order means that we never have to back the $p^c(\cdot)$ pointer up: the search for $V(<p^c(e),n>,|N=m)$ can start where the pointer was last left. Similar observations hold for the $p^{\bar{c}}(\cdot)$ pointer. Since $C_m(<\cdot,n-1>|N=m)$ is linear, it is easy to compare $C_m(<e,n-1>|N=m)$ and $C_t(<e,n-1>|N=m)$ for every C_t transition point e. When we encounter the first e such that $C_m(<e,n-1>|N=m) < C_t(<e,n-1>|N=m)$, we know that C_t and C_m intersect in the interval between e and the greatest transition point less than e. It is then straightforward to calculate the point of intersection, so that the complete set of transition points for $V(<\cdot,n-1>|N=m)$ are defined. The slopes and intercepts for each of the V's linear segments are then calculated. Each of the steps outlined above has linear complexity in twice the size of $D(V(<\cdot,n>|N=m))$.

The procedure outlined so far will calculate the optimal cost function exactly. But because the number of transition points essentially double at every step, the computational complexity and storage requirements of the algorithm are prohibitive. We next describe a method of constructing an accurate approximation to $V(<\cdot,n>|N=m)$ which significantly reduces the number of transition points required to represent it. We assume that $V(<\cdot,n>|N=m)$ or an approximation to it is known, and is denoted by W. We assume that W is piece-wise linear and concave. We also assume an error tolerance ε, and desire an approximation to W which bounds the maximal error between W and its approximation, which minimizes the number of transition points used to achieve that tolerance, and which remains concave. We will restrict ourselves to *interior approximations* defined as follows. If A is an interior approximation to W, then $A(e) = W(e)$ at each of A's transition points, A is piece-wise linear, and

$A(p) \leq W(p)$ for all p. If W is concave then A will be concave. Our approximation procedure finds an interior approximation to W which minimizes over all interior approximations the number of transition points needed to achieve $W(p) - A(p) \leq \varepsilon$ for all p.

The basic idea of the approximation is to build up its line segments piece by piece. Starting with the left most endpoint of the interval ($p = 0$), the rightmost endpoint of A's first segment is chosen so that the maximal difference between A and W over that interval is exactly ε. The chosen endpoint is then taken as the left endpoint of the next approximation segment, and again the right endpoint is chosen so that the maximal error of the second approximation segment is exactly ε. This process is repeated until W is completely approximated. The number of transition points chosen using this method minimizes the number of transition points needed by an interior approximation to achieve an error tolerance of ε. Furthermore, the complexity of approximating W by A is linear in the number of W's transition points. We now outline this approximation in more detail.

Suppose that l is the left endpoint of the interval for A that we are attempting to construct, and consider the interior approximation to W consisting of a single linear segment between $W(l)$ and $W(u)$, $u > l$. To emphasis the right endpoint, we will call this segment the u-segment. The point at which the difference between W and the u-segment is maximized over $[l, u]$ is the right hand endpoint associated with W's linear segment having the least slope greater than the u-segment slope, $(W(u) - W(l))/(u - l)$. Furthermore, the maximal error between the u-segment and W over $[l, u]$ is a continuous increasing function of u. Thus, given l, we can calculate the maximal error over $[l, d_i]$, for any of W's transition points $d_i > l$, the d_i being examined in increasing sorted order. Upon finding the first d_j such that the maximal error between W and the d_j-segment exceeds ε, we can calculate the point u', $d_{j-1} \leq u' \leq d_j$ such that the u'-segment's maximal error over $[l, u']$ is exactly ε, as follows. Let m and b be the slope and intercept of the W segment between $W(d_{j-1})$ and $W(d_j)$. For $u \in [d_{j-1},d_j]$ the slope of the u-segment is

$$m(u) = \frac{m \cdot u + b - W(l)}{u - l}$$

and its intercept is

$$b(u) = W(l) - m(u) \cdot l.$$

At any point $e \in [l, u]$, the error between W and the u-segment is given by

$$W(e) - \left[m(u) \cdot e + b(u)\right]. \tag{7}$$

Let $e(u)$ be the W transition point at which the u-segment's maximal error over $[l, u]$ occurs. For $u \in [d_{j-1}, d_j]$ $e(u)$ is an increasing step function; let $u_1, \ldots, u_k$ be the ordered sequence of points where $e(u)$ is discontinuous, and $u_1 = d_{j-1}$. It will be understood that $e(u_i)$ denotes the value of $e(\cdot)$ as u_i is approached from the right. Given $e(u_i)$, it is not difficult to identify u_{i+1}. We know that $e(\cdot)$ takes a step up precisely when the slope of the u-segment is equal to the slope of the W segment between $e(u_i)$ and $e(u_{i+1})$; since this slope m_i is known from W's description, and it is known that u-segment passes through $(l, W(l))$ regardless of u, it is a simple matter to calculate where a line with slope m_i which passes through $(l, W(l))$ will intersect the W segment between $W(d_{j-1})$ and $W(d_j)$. The point of intersection will be u_{i+1}.

$e(u_1)$ was identified when the interval $[d_{j-1}, d_j]$ was identified, so that $e(u_2)$ is the next larger W transition point, and u_2 can be identified as outlined above. Now we hypothesize that the u which leads to ε maximum error is in the interval $[d_{j-1}, u_2]$, so that the maximum error occurs at $e(u_1)$. Following expression (7), we solve for u in the equation

$$\varepsilon = W(e(u_1)) - \left[m(u) \cdot e(u_1) + b(u)\right].$$

If the solution u' is less than u_2, then we select u' as the approximation endpoint. Otherwise, we hypothesize that the endpoint sought is in the interval $[u_2, u_3]$, and solve for u in the equation

$$\varepsilon = W(e(u_2)) - \left[m(u) \cdot e(u_2) + b(u)\right].$$

Again, $e(u_3)$ is the next larger transition point from $e(u_2)$, and if the solution u' to the equation is less than u_3 we are done. This process is repeated until the u' which causes the maximal error over $[l, u']$

to be exactly ε is found. u' then defines the right endpoint of the A segment under construction. u' is then used as the left endpoint for the next segment, found using this same procedure. Because every one of W's transition points is scanned at most once in looking for a bounding interval on u', and at most once in narrowing the search for u' in that bounding interval, the complexity of approximating W with A is linear in the number of W's transition points.

If W is an approximation to $V(<\cdot,n>|N=m)$ whose error is no larger than δ, and if W's approximation A has error no larger than ε, then A is an approximation to $V(<\cdot,n>|N=m)$ with error no larger than $\delta + \varepsilon$. Thus we can bound the error on $V(<\cdot,1>|N=m)$ by ε if we construct approximations to each $V(<\cdot,n>|N=m)$ with tolerance ε/m. In practice, we achieve somewhat better accuracy by calling upon the approximation technique only when the number of transition points for the working level of $V(<\cdot,n>|N=m)$ gets large. The approximation typically reduces the number of transition points significantly, allowing us to do an exact mapping until the number of points again grows too large.

The procedures discussed above initially condition on $N = m$. We suppose then that $C_t(<\cdot,n>|N=m)$ and $V(<\cdot,n>|N=m)$ have been calculated or approximated for every m such that $Prob\{N = m\} \neq 0$. Combining the conditional functions is relatively straightforward. The transition points for $C_t(<\cdot,n>)$ are found by merging the transition points for each conditional function $C_t(<\cdot,n>|N=m)$. Then for every $e \in D(Ct(<\cdot,n>))$, we calculate the function value

$$C_t(<e,n>) = \sum_{m=1}^{M} Prob\{N = m\} C_t(<e,n>|N=m).$$

Note that if the approximation to $C_t(<e,n>|N=m)$ has maximal error of ε, then $Prob\{N = m\}$ times that approximation deviates from the true product $Prob\{N = m\}C_t(<e,n>|N=m)$ by no more than $\varepsilon \cdot Prob\{N = m\}$. It follows that if for every m we approximate $C_t(<e,n>|N=m)$ with tolerance ε, then the sum above has tolerance ε.

Since $C_m(<\cdot,n>|N=m)$ is linear for every m, $C_m(<\cdot,n>)$ is also linear; its slope and intercept can be found by evaluating $C_m(<0,n>)$ and $C_m(<1,n>)$ as above. The intersection of $C_m(<\cdot,n>)$ and

$C_i(< \cdot, n>)$ can then be determined, and π_n discovered.

References

[1] I. Babuska , Ed. *Adaptive Computational Methods for Partial Differential Equations*, SIAM, Philadelphia, 1983.

[2] M.J. Berger and S. Bokhari, The Partitioning of Non-Uniform Problems,*ICASE Report No. 85-55*, November 1985.

[3] S. Bokhari, Partitioning Problems in Parallel, Pipelined, and Distributed Computing, *ICASE Report No. 85-54*, November 1985.

[4] G. Browning, H.-O. Kreiss, J Oliger, Mesh Refinement, *Mathematics of Computation*, Vol. 27, (1973), pp 29-39.

[5] W.W. Chu, L.J. Holloway, M. Lan, and K. Efe, Task Allocation in Distributed Data Processing, *Computer, 13, 11*, November 1980, 57-69.

[6] Y. Chow and W. Kowhler, Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System, *IEEE Trans. on Computers, C-28, 5*, (May 1979), 354-361.

[7] T. C. Chou and J. A. Abraham, Load Balancing in Distributed Systems, *IEEE Trans. on Software Eng., 8*, 4 (July 1982), 401-412.

[8] D.L. Eager, E.D. Lazowska, J. Zahorjan, Adaptive Load Sharing in Homogeneous Distributed Systems, *IEEE Trans. on Software Engineering*, Vol SE-12. No. 5, (May 1986), pp 662-675.

[9] G. J. Foschini, On Heavy Traffic Diffusion Analysis and Dynamic Routing in Packet Switched Networks, in *Computer Performance*, K. M. Chandy and M. Reiser Eds. New York: North-Holland, 1977.

[10] D. Gusfield, Parametric Combinatorial Computing and a Problem of Program Module Distribution, *Journal of the ACM, 30, 3*, July 1983, 551-563.

[11] D.I. Moldovan and J.A.B. Fortes, Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays, *IEEE Trans. on Computers, C-35, 1*, (January 1986), 1-12.

[12] P.R. Ma, E.Y.S. Lee, and M. Tsuchiya, A Task Allocation Model for Distributed Computing Systems, *IEEE Trans. on Computers, C-31, 1*, January 1982, 41-47.

[13] L. M. Ni, C. Xu, and T.B. Gendreau, A Distributed Drafting Algorithm for Load Balancing, *IEEE Trans. on Software Engineering, SE-11, 10*, October 1985, 1153-1161.

[14] D. M. Nicol and P. F. Reynolds, Jr., The Automated Partitioning of Simulations for Parallel Execution, *University of Virginia Department of Computer Science Tech Report TR-85-15*, August 1985.

[15] D. M. Nicol and P. F. Reynolds, Jr., An Optimal Repartitioning Decision Policy, *ICASE Report No. 86-7*, Feb., 1986.

[16] D. M. Nicol, J. H. Saltz, Dynamic Remapping of Parallel Computations With Varying Resource Demands, *ICASE Report No. 86-45, July 1986.*

[17] C.C. Price, U.W. Pooch, Search Techniques for a Nonlinear Multiprocessor Scheduling Problem, *Naval Research Logistics Quarterly, 29, 2*, June 1982, 213-233.

[18] A. Rapoport, W. E. Stein, and G. J. Burkheimer, *Response Models for Detection of Change*, D. Reidel Publishing Company, Boston, 1979.

[19] S. Ross, *Applied Probability Models with Optimization Applications*, Holden-Day, San Francisco, 1970.

[20] S. Ross, *Stochastic Processes*, Wiley and Sons, New York, 1983.

[21] S. A. Schmitt, *An Elementary Introduction to Bayesian Statistics*, Addison-Wesley, 1969.

[22] J. A. Stankovic, An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling, *IEEE Trans. on Computers, C-34*, 2 (Feb 1985), 117-130.

[23] J. A. Stankovic, K. Ramamritham and S. Cheng, Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems, *IEEE Trans. on Computers, C-34, 12 (December 1985), 1130-1143.*

[24] H.S. Stone, Critical Load Factors in Distributed Computer Systems, *IEEE Trans. on Software Engineering, SE-4, 3* (May 1978), 254-258.

[25] D. Towsley, Queueing Network Models with State-Dependent Routing, *Journal of the ACM*, 27, 2 (April 1980) 323-337.

[26] A. N. Tantawi and D. Towsley, Optimal Static Load Balancing, *Journal of the ACM, 32*, 2 (April 1985), 445-465.

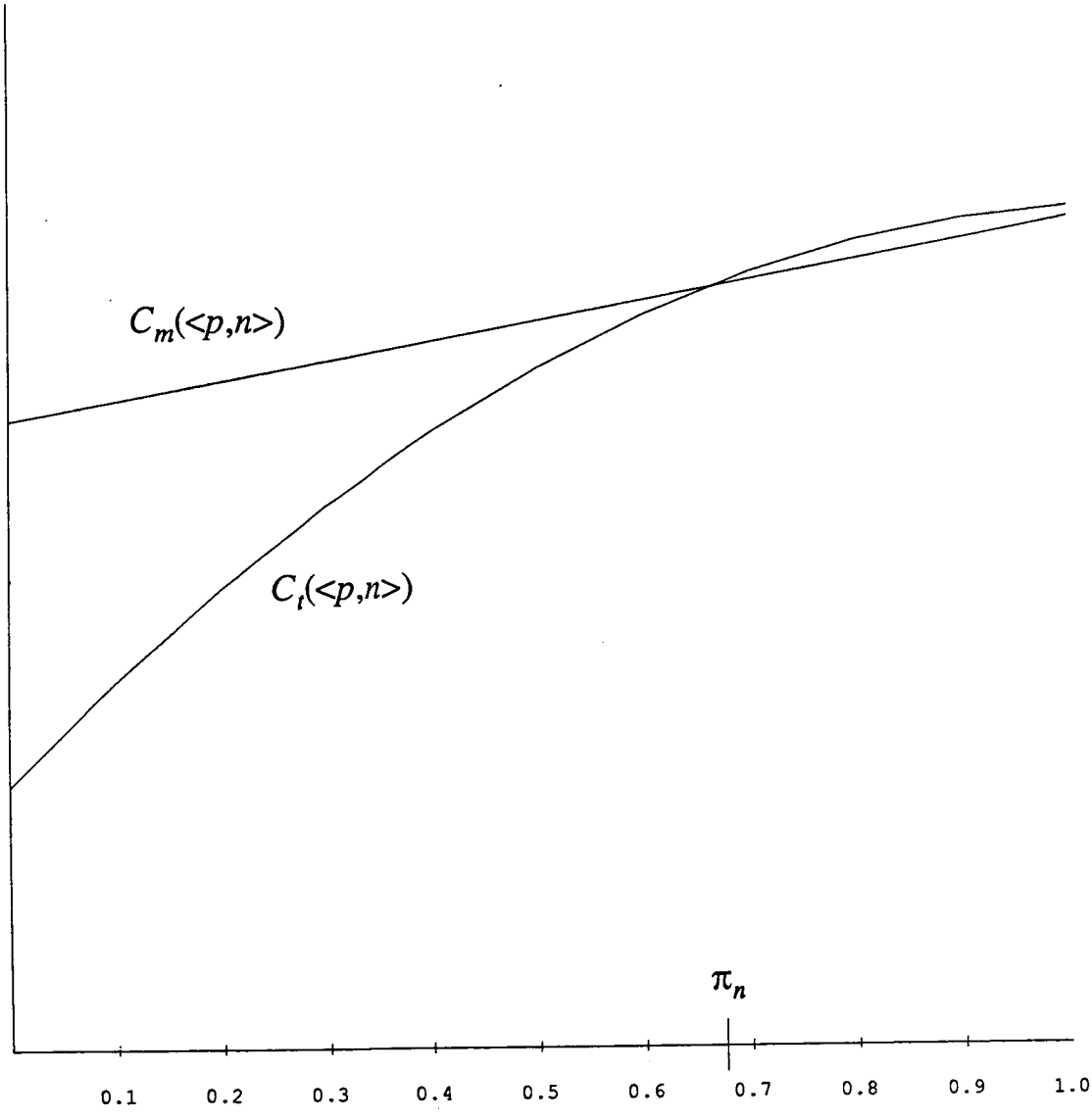


Fig. 1 *Intersection of* $C_t(<p,n>)$ and $C_m(<p,n>)$

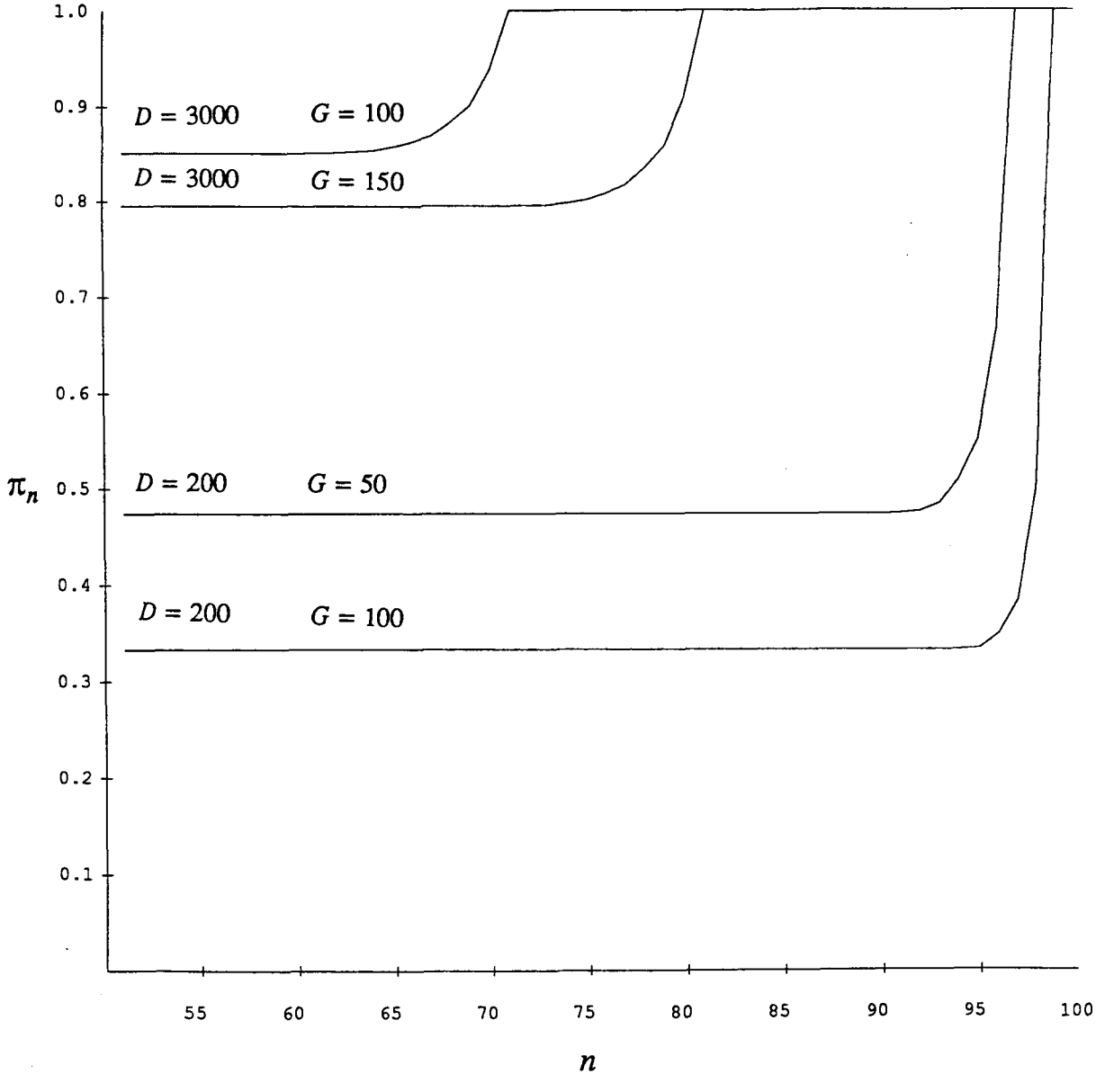


Fig. 2: *Behavior of* π_n

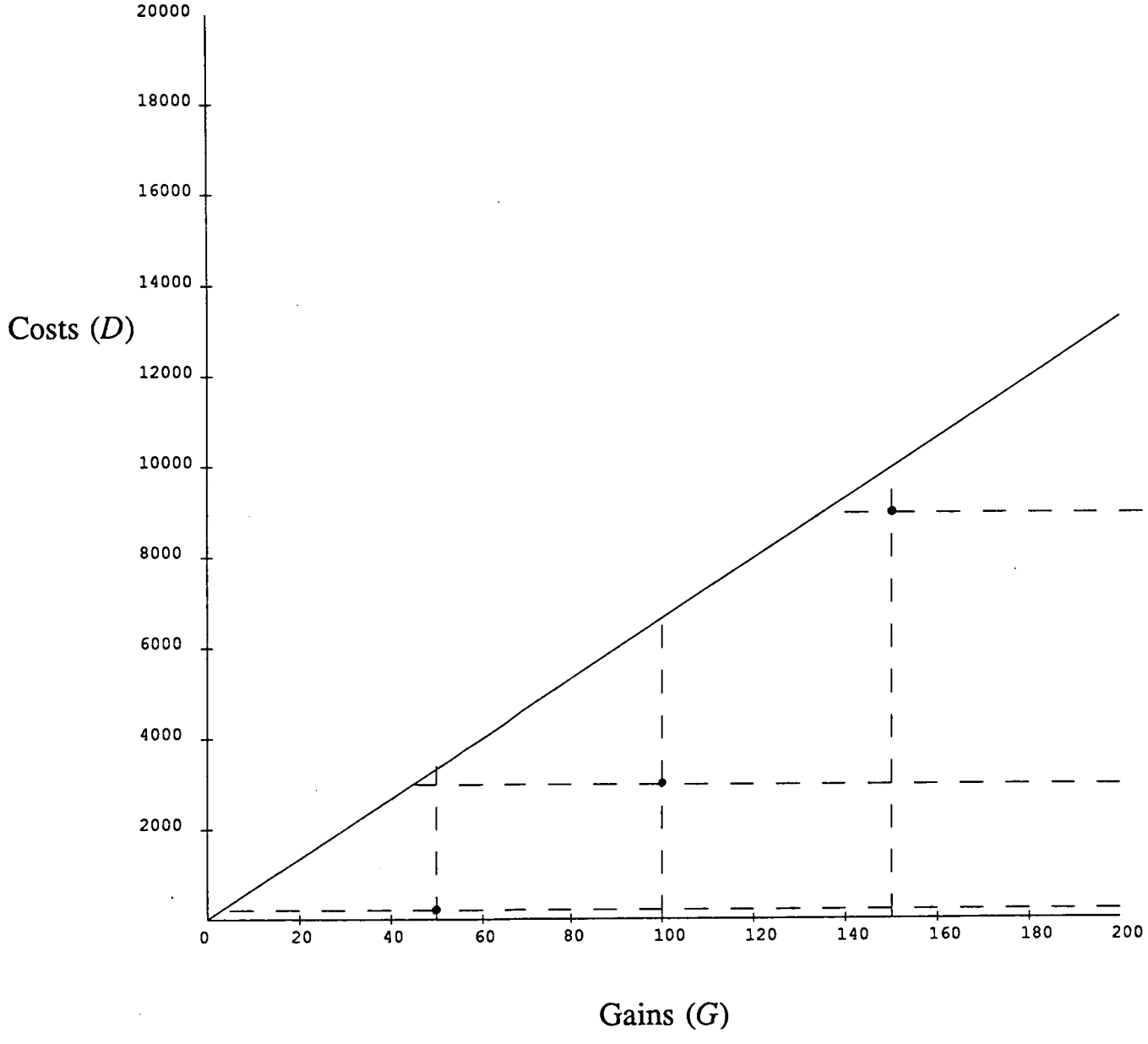


Fig. 3 *Envelope of Admissible Costs and Gains*

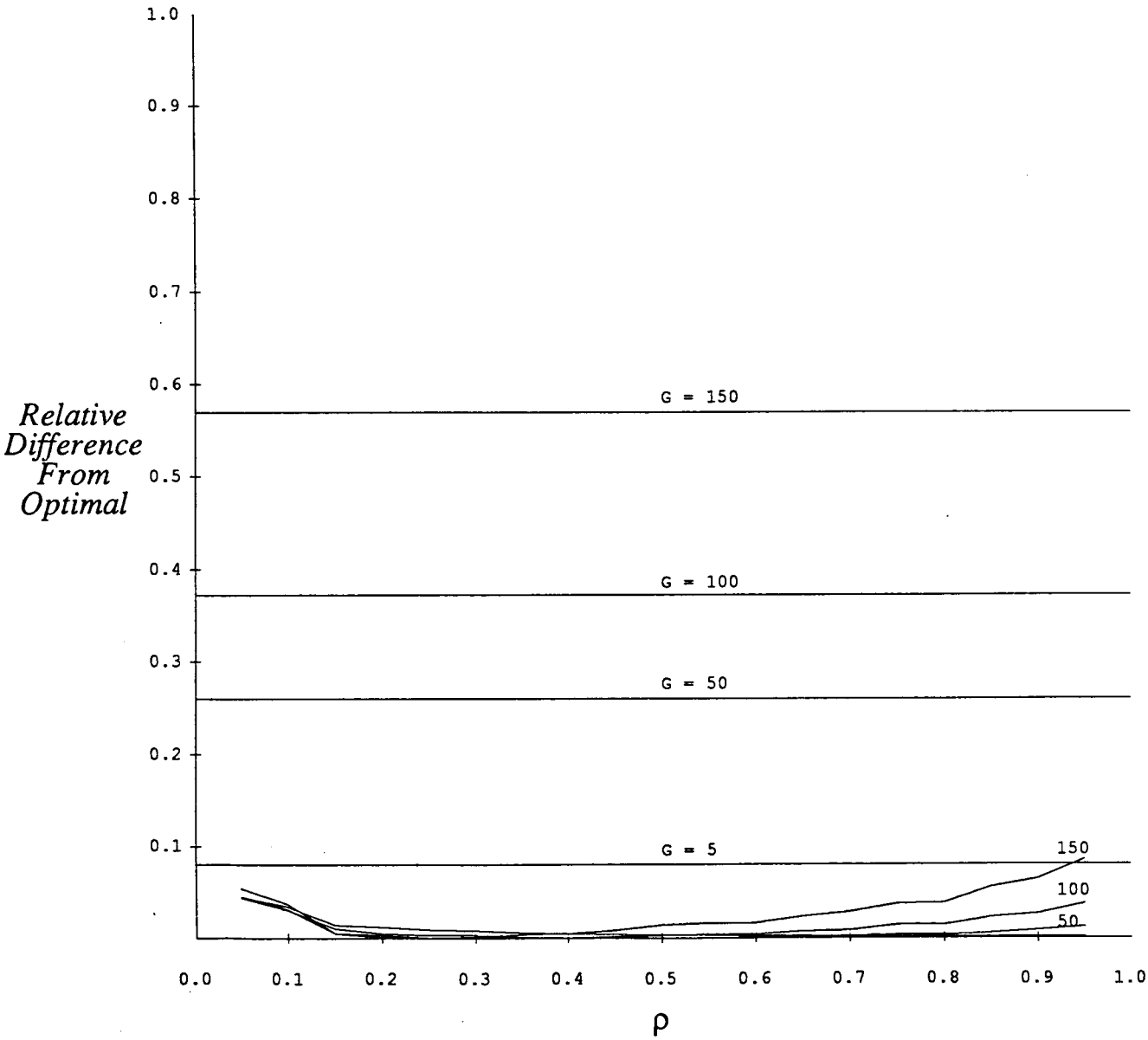


Fig. 4 *Sensitivity to G When D = 200*

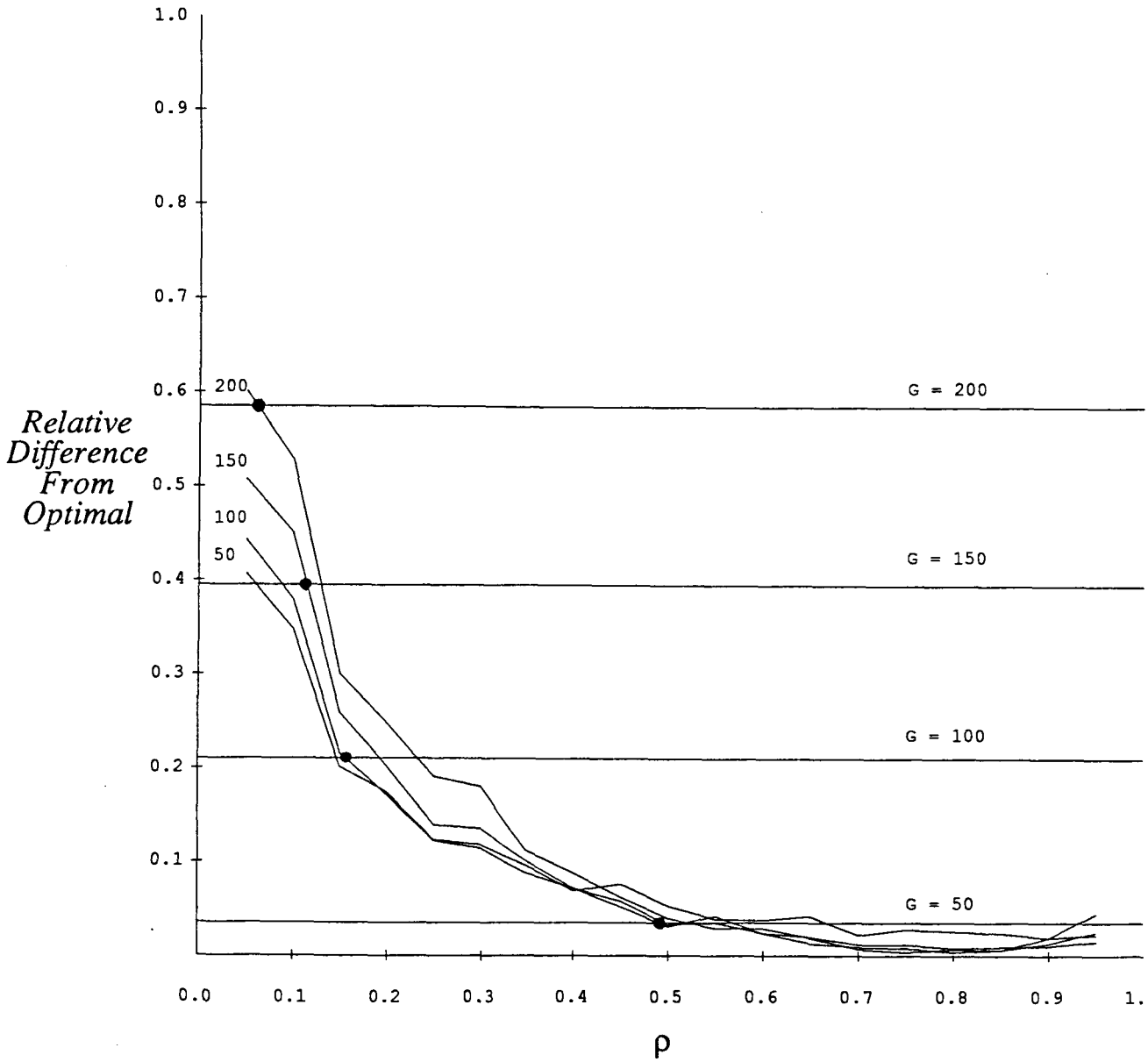


Fig. 5 *Sensitivity to G When D* = 3000

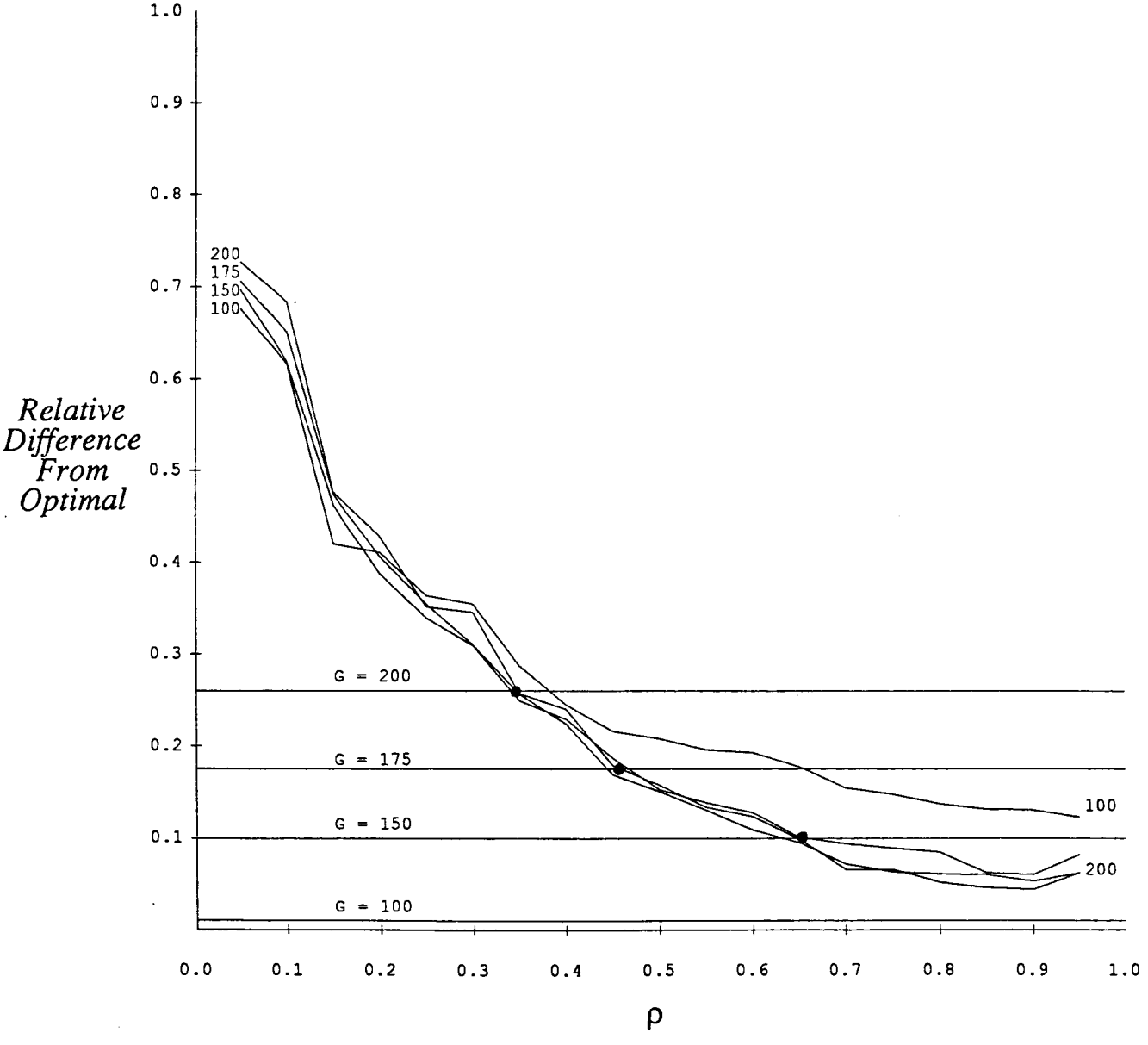


Fig. 6 *Sensitivity to G When D = 9000*

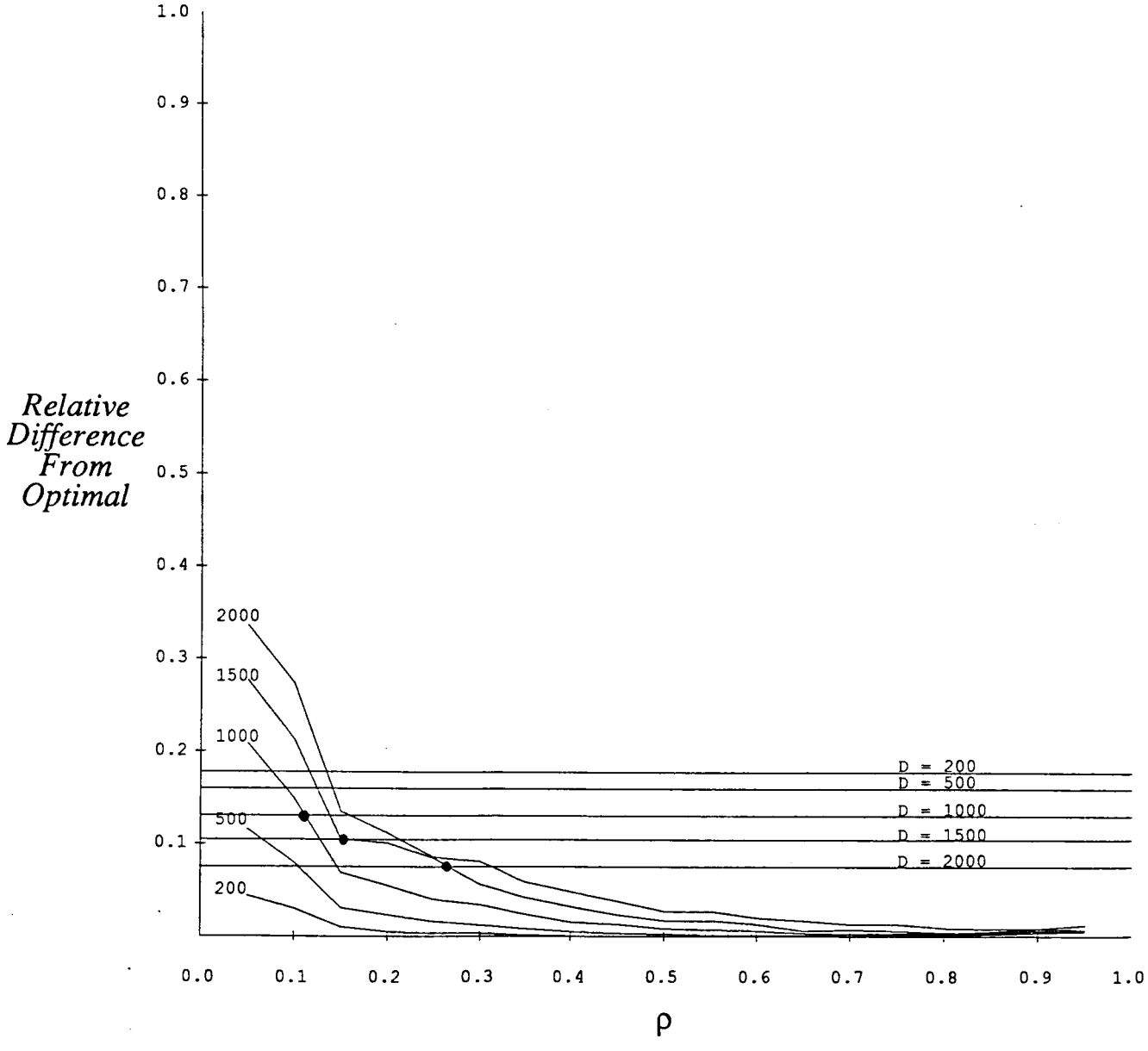


Fig. 7 *Sensitivity to D When G = 50*

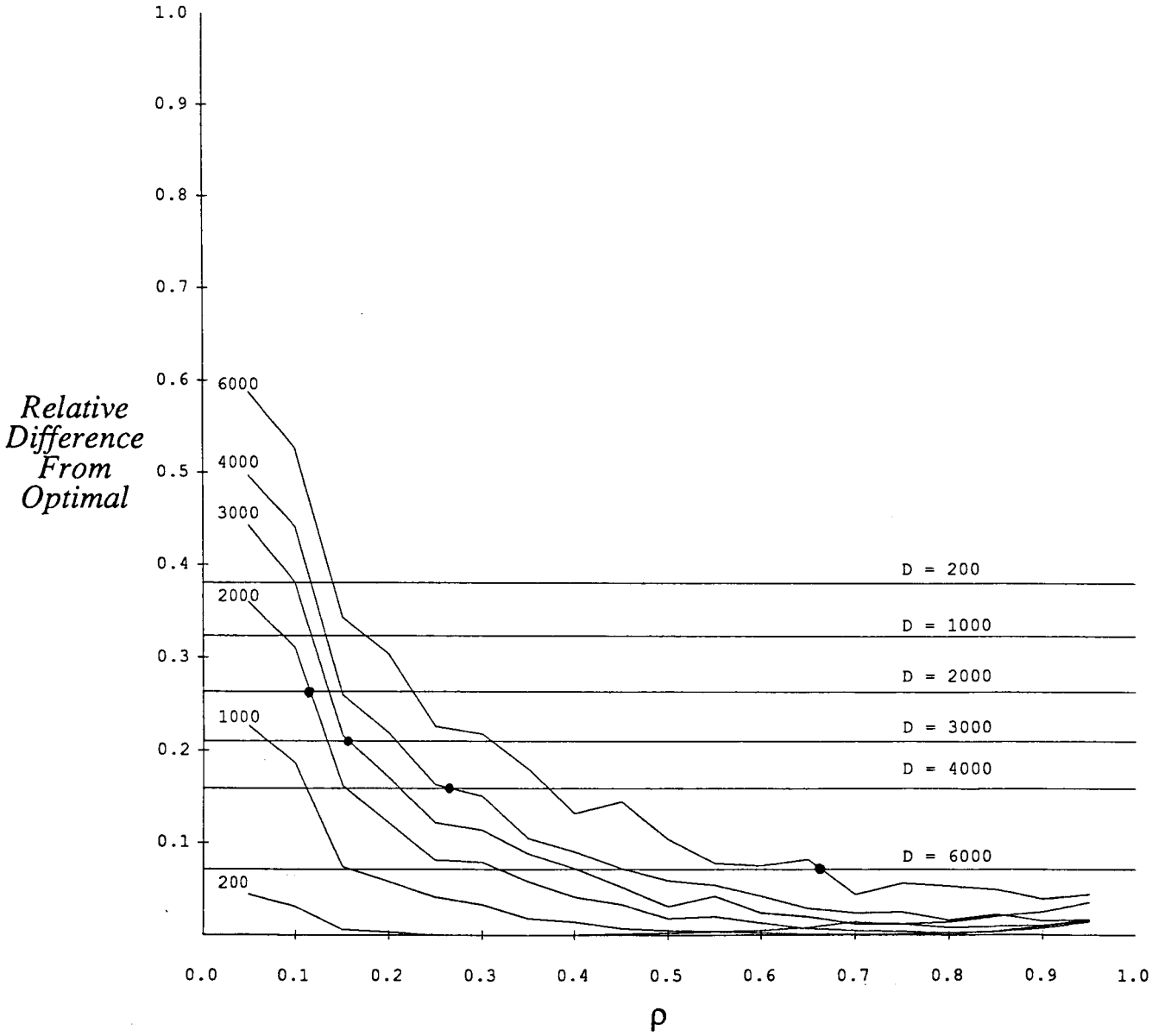


Fig. 8 *Sensitivity to D When G = 100*

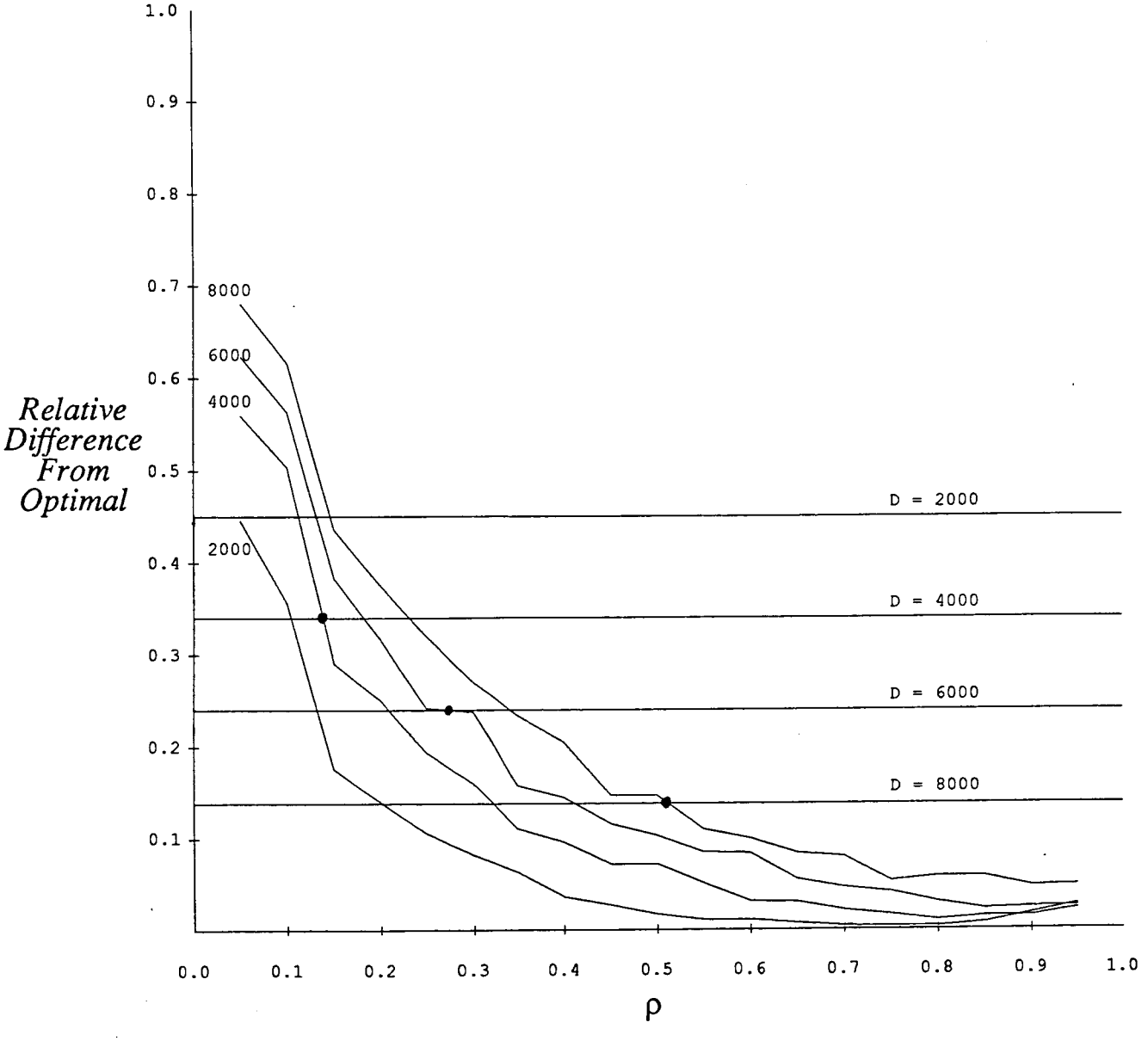


Fig. 9 *Sensitivity to D When G = 150*

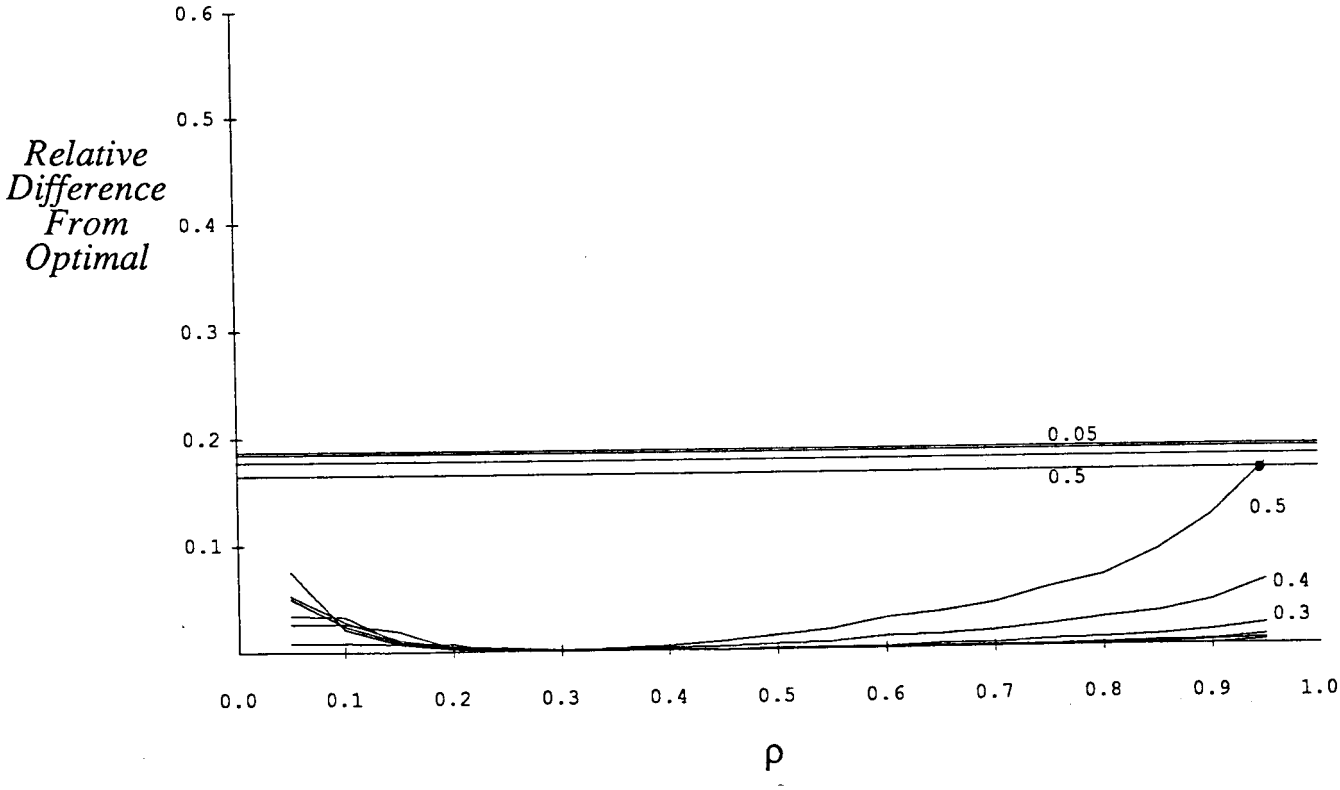


Fig. 10 *Sensitivity to* α, β *When* $G = 50$, $D = 200$

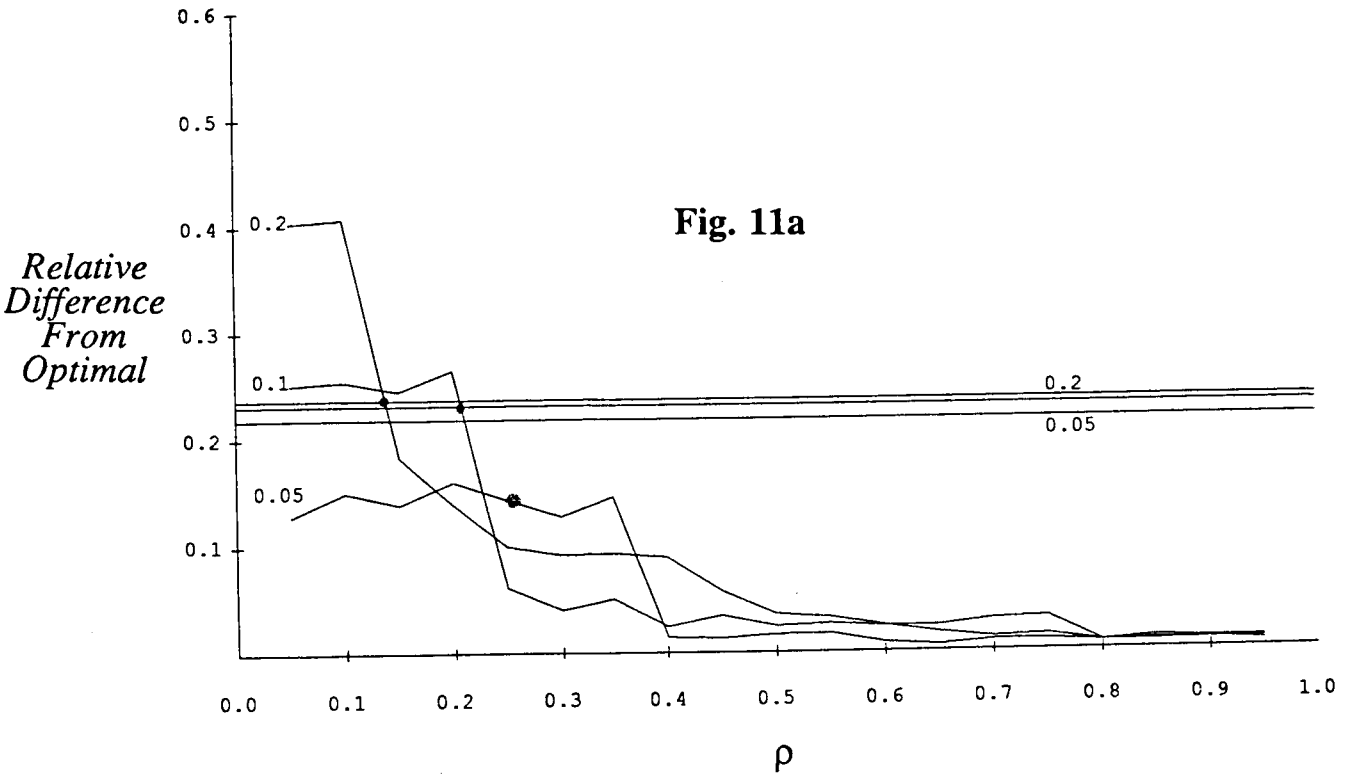


Fig. 11a

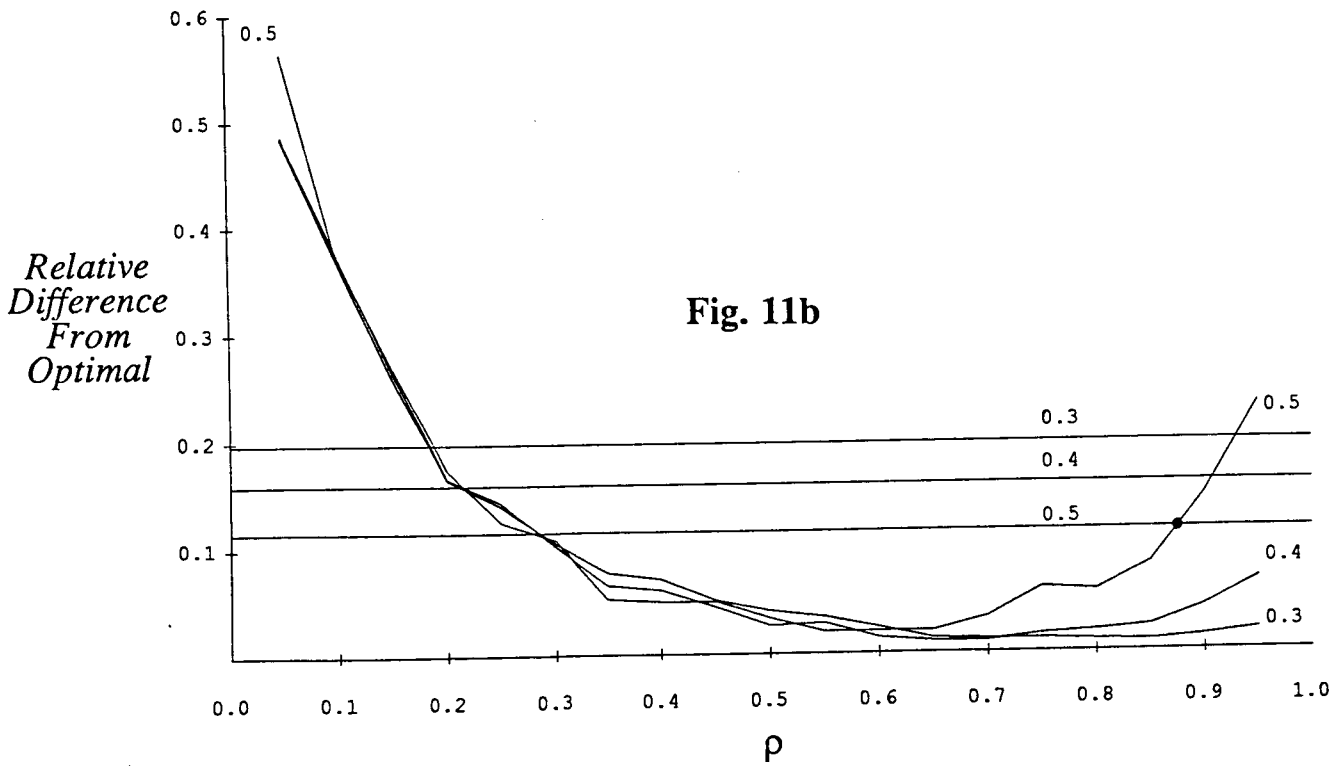
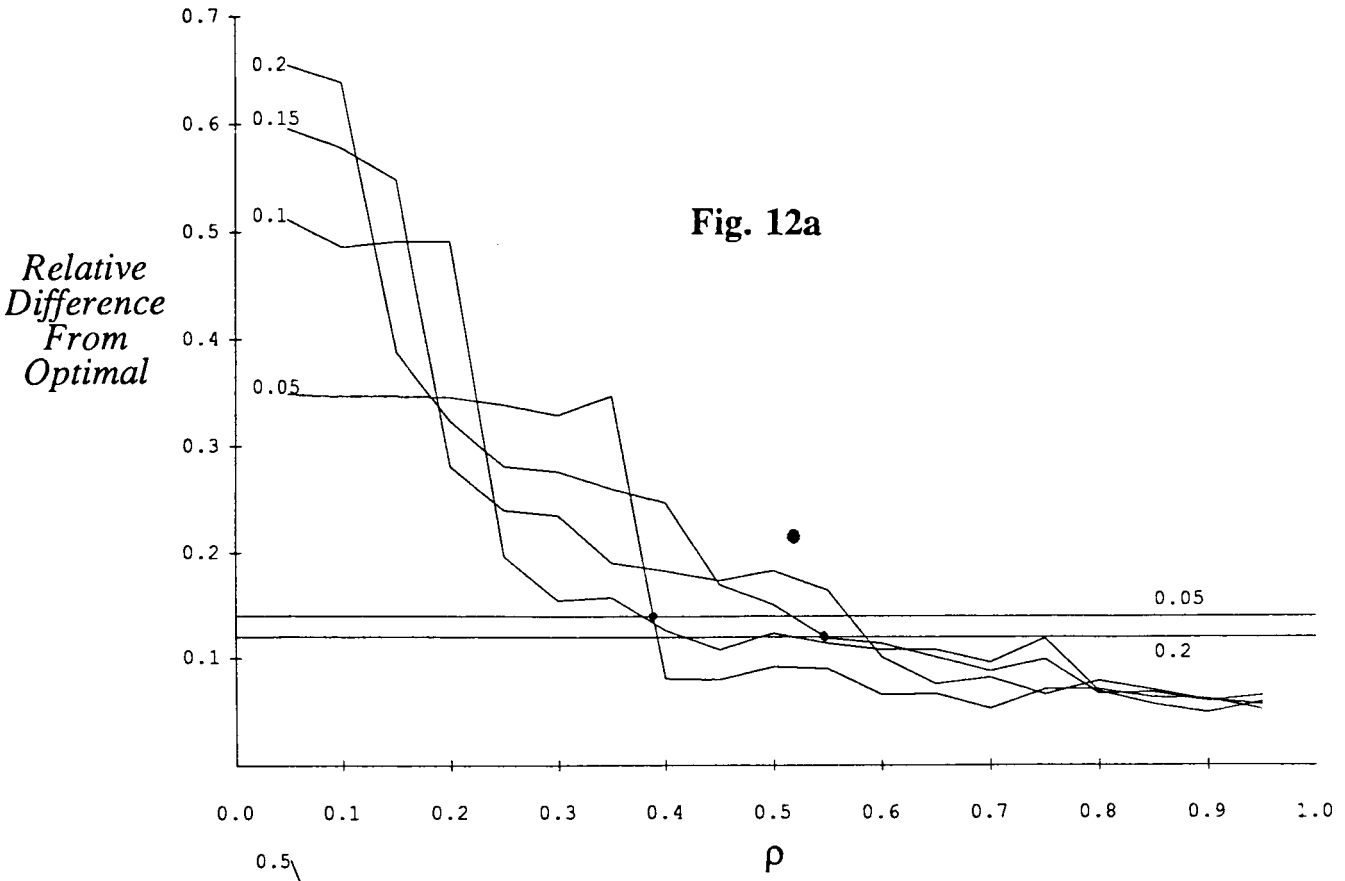


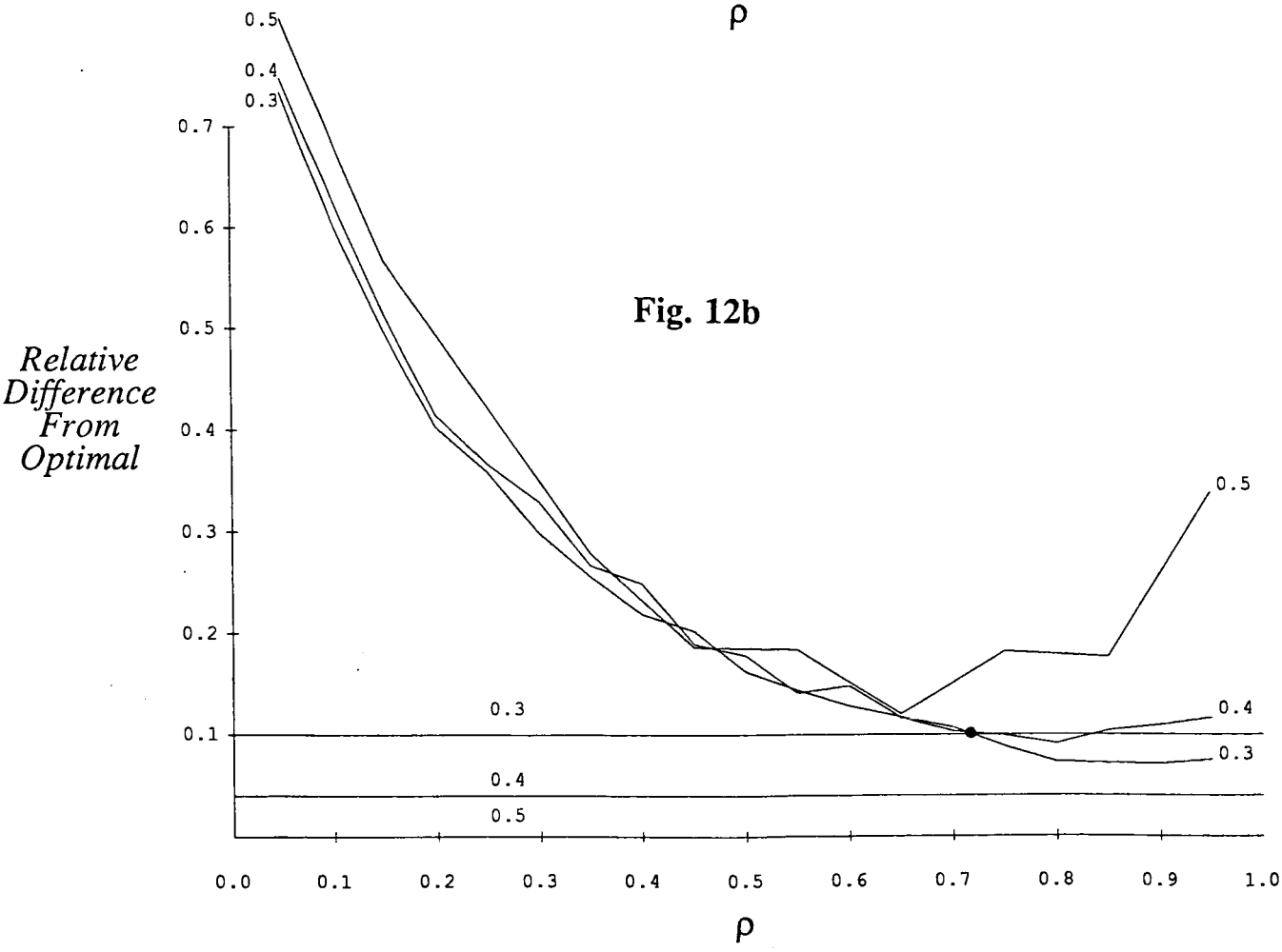
Fig. 11b

Fig. 11 *Sensitivity to* α, β *When* $G = 100$, $D = 3000$



Relative Difference From Optimal

Fig. 12a



Relative Difference From Optimal

Fig. 12b

Fig. 12 *Sensitivity to* α, β *When* $G = 150$, $D = 9000$

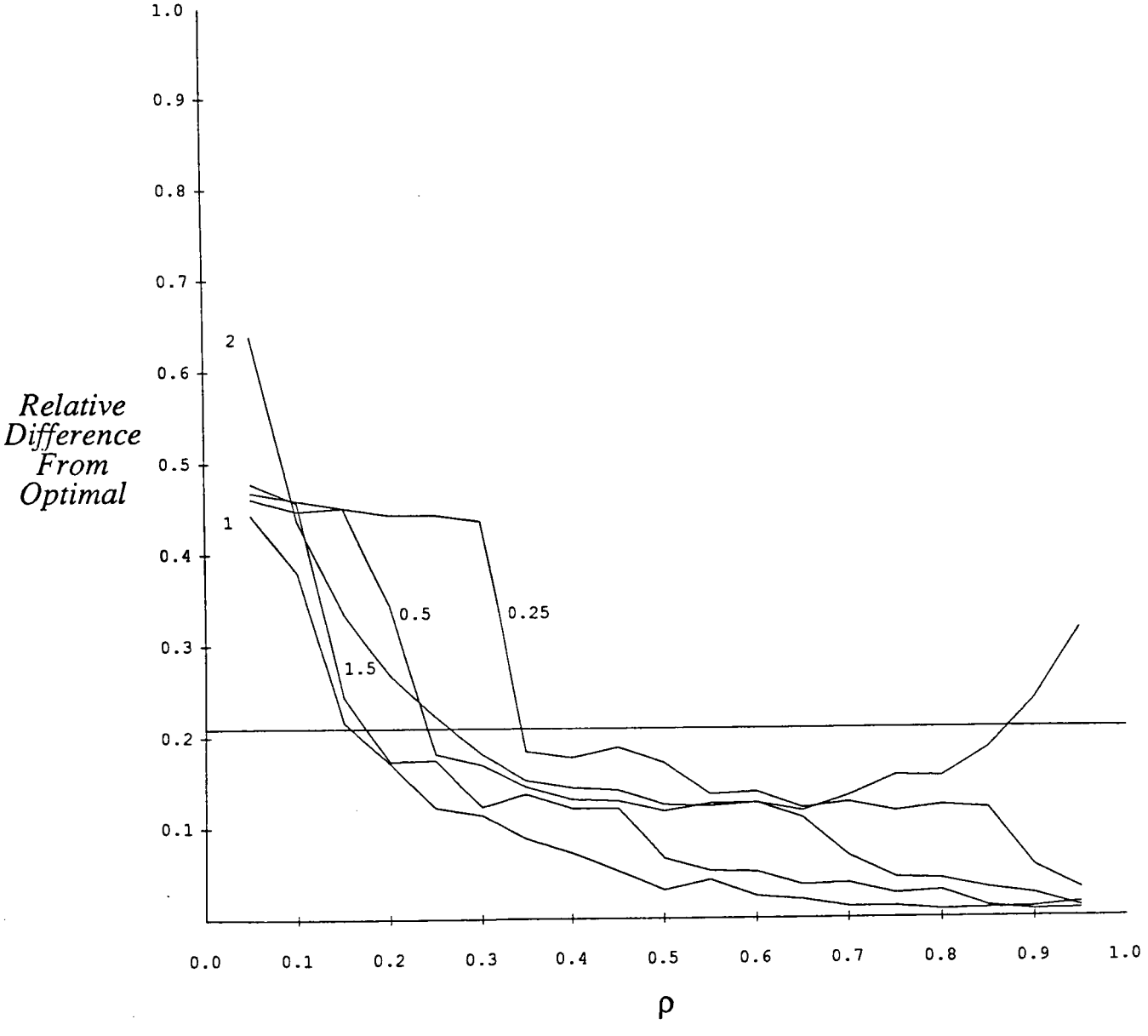


Fig. 13 *Sensitivity to Mis-estimation of* α, β

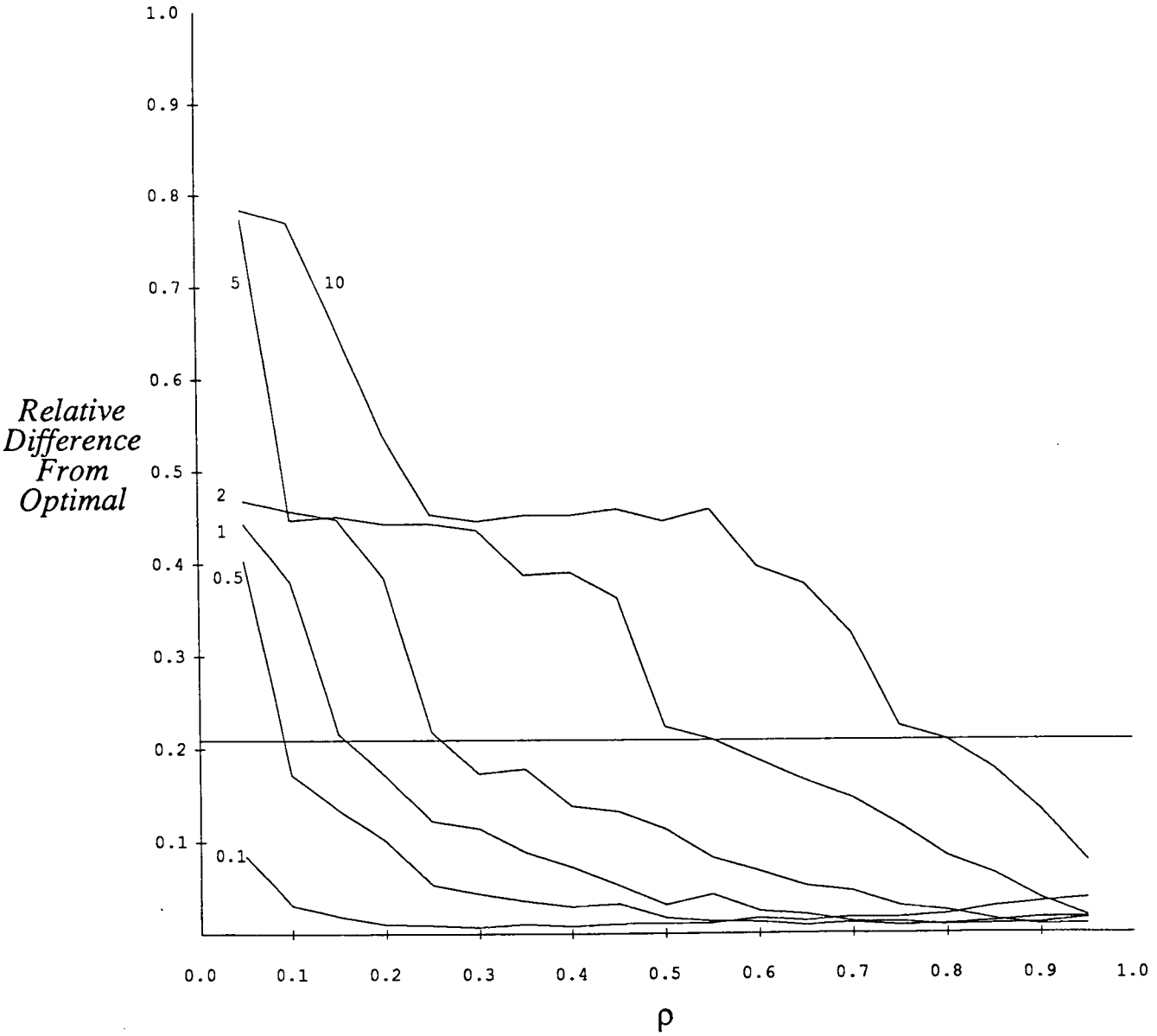


Fig. 14 *Sensitivity to Mis-estimation of* ϕ

Standard Bibliographic Page

1. Report No. NASA CR-178174 ICASE Report No. 86-58	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle DYNAMIC REMAPPING DECISIONS IN MULTI-PHASE PARALLEL COMPUTATIONS		5. Report Date September 1986 6. Performing Organization Code
7. Author(s) David M. Nicol and Paul F. Reynolds, Jr.		8. Performing Organization Report No. 86-58
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225		10. Work Unit No. 11. Contract or Grant No. NAS1-17070, NAS1-18107
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Contractor Report 14. Sponsoring Agency Code 505-90-21-01

15. Supplementary Notes

Langley Technical Monitor:
J. C. South

Submitted to IEEE Trans. on Computers

Final Report

16. Abstract

The effectiveness of any given mapping of workload to processors in a parallel system is dependent on the stochastic behavior of the workload. Program behavior is often characterized by a sequence of phases, with phase changes occurring unpredictably. During a phase, the behavior is fairly stable, but may become quite different during the next phase. Thus a workload assignment generated for one phase may hinder performance during the next phase. We consider the problem of deciding whether to remap a parallel computation in the face of uncertainity in remapping's utility. Fundamentally, it is necessary to balance the expected remapping performance gain against the delay cost of remapping. This paper treats this problem formally by constructing a probabilistic model of a computation with at most two phases. We use stochastic dynamic programming to show that the remapping decision policy which minimizes the expected running time of the computation has an extremely simple structure: the optimal decision at any step is followed by comparing the probability of remapping gain against a threshold. This theoretical result stresses the importance of detecting a phase change, and assessing the possibility of gain from remapping. We also empirically study the sensitivity of optimal performance to imprecise decision thresholds. Under a wide range of model parameter values, we find nearly optimal performance if remapping is chosen simply when the gain probability is high. These results strongly suggest that except in extreme cases, the remapping decision problem is essentially that of dynamically determining whether gain can be achieved by remapping after a phase change; precise quantification of the decision model parameters is not necessary.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
parallel processing, multi-processors, load balancing	61 - Computer Programming and Software 66 - Systems Analysis Unclassified - unlimited

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	47	A03

For sale by the National Technical Information Service, Springfield, Virginia 22161